SIAM J. COMPUT. © 2004 Society for Industrial and Applied Mathematics
Vol. 34, No. 1, pp. 1–22

# FAST UNIVERSALIZATION OF INVESTMENT STRATEGIES[*]

KARHAN AKCOGLU[†], PETROS DRINEAS[‡], AND MING-YANG KAO[§]

**Abstract.** A *universalization* of a parameterized investment strategy is an online algorithm whose average daily performance approaches that of the strategy operating with the optimal parameters determined offline in hindsight. We present a general framework for universalizing investment strategies and discuss conditions under which investment strategies are universalizable. We present examples of common investment strategies that fit into our framework. The examples include both trading strategies that decide positions in individual stocks, and portfolio strategies that allocate wealth among multiple stocks. This work extends in a natural way Cover's universal portfolio work. We also discuss the runtime efficiency of universalization algorithms. While a straightforward implementation of our algorithms runs in time exponential in the number of parameters, we show that the efficient universal portfolio computation technique of Kalai and Vempala [*Proceedings of the 41st Annual IEEE Symposium on Foundations of Computer Science*, Redondo Beach, CA, 2000, pp. 486–491] involving the sampling of log-concave functions can be generalized to other classes of investment strategies, thus yielding provably good approximation algorithms in our framework.

**Key words.** universal portfolios, computational finance, portfolio optimization, investment strategies, portfolio strategies, trading strategies, constantly rebalanced portfolios

**AMS subject classifications.** 91B28, 68Q32, 68W40

**DOI.** 10.1137/S0097539702405619

## 1. Introduction.

An age-old question in finance deals with how to manage money on the stock market to obtain an "acceptable" return on investment. An *investment strategy* is an online algorithm that attempts to address this question by applying a given set of rules to determine how to invest capital. Typically, an investment strategy is parameterized by a vector $\mathbf{w} \in \mathbb{R}^* = \bigcup_{i=1}^{\infty} \mathbb{R}^i$ that dictates how the strategy operates. The optimal parameters that maximize the strategy's return are unknown when the algorithm is run, and the parameters are usually chosen quite arbitrarily. A *universalization* of an investment strategy is an online algorithm based on the strategy whose average daily performance approaches that of the strategy operating with the optimal parameters determined offline in hindsight.

Consider the *constantly rebalanced portfolio* (CRP) investment strategy universalized by Cover [5] and the subject of several extensions and generalizations [3, 6, 11, 14, 16, 20]. The CRP strategy maintains a constant proportion of total wealth in each stock, where the proportions are dictated by the parameters given to the strategy. In

[*]Received by the editors April 12, 2002; accepted for publication (in revised form) March 27, 2004; published electronically October 1, 2004. A preliminary version of this work appeared in *Lecture Notes in Computer Science* 2380: *Proceedings of the* 29th *International Colloquium on Automata, Languages, and Programming*, M. Hennessy and P. Widmayer, eds., Springer-Verlag, New York, 2002, pp. 888–900.

http://www.siam.org/journals/sicomp/34-1/40561.html

[†]The Goldman Sachs Group, New York, NY 10004 (karhan.akcoglu@aya.yale.edu). This work was done while the author was a graduate student at Yale University and was supported in part by NSF grant CCR-9988376.

[‡]Department of Computer Science, Rensselaer Polytechnic Institute, Troy, NY 12180 (drinep@cs.rpi.edu). This work was done while the author was a graduate student at Yale University and was supported in part by NSF grant CCR-9896165.

[§]Department of Computer Science, Northwestern University, Evanston, IL 60201 (kao@cs.northwestern.edu). This author's work was supported in part by NSF grant CCR-9988376.

a stock market with $m$ stocks, the parameter space for the CRP strategy is

$$\mathcal{W}_m = \left\{ \mathbf{w} \in [0,1]^m \mid \sum_{i=1}^{m} w_i = 1 \right\},$$

the set of vectors in $\mathbb{R}^m$ whose components are between 0 and 1 and add up to 1. Given a *portfolio vector* $\mathbf{w} = (w_1, \ldots, w_m) \in \mathcal{W}_m$, $w_i$ tells us the proportion of wealth to invest in stock $i$ for $1 \leq i \leq m$. At the beginning of each day, the holdings are *rebalanced*; i.e., money is taken out of some stocks and put into others, so that the desired proportions are maintained in each stock. As an example of the robustness of the CRP strategy, consider the following market with two stocks [11, 16]. The price of one stock remains constant, while the other stock doubles and halves in price on alternate days. Investing in a single stock will at most double our money. With a CRP $(\frac{1}{2}, \frac{1}{2})$ strategy, however, our wealth will increase exponentially, by a factor of $(\frac{1}{2} \cdot 1 + \frac{1}{2} \cdot 2) \times (\frac{1}{2} \cdot 1 + \frac{1}{2} \cdot \frac{1}{2}) = \frac{3}{2} \times \frac{3}{4} = \frac{9}{8}$ every two days.

Cover developed an investment strategy that effectively distributes wealth uniformly over all portfolio vectors $\mathbf{w} \in \mathcal{W}_m$ on the first day and executes the CRP strategy with daily rebalancing according to each $\mathbf{w}$ on the (infinitesimally small) proportion of wealth initially allocated to each $\mathbf{w}$. Cover showed that the *average daily log-performance*[1] of such a strategy approaches that of the CRP strategy operating with the optimal return-maximizing parameters chosen with hindsight.

This paper generalizes previous results and introduces a framework that allows universalizations of other parameterized investment strategies. As we see in section 2, investment strategies typically fall under two categories: *trading strategies* operate on a single stock and dictate when to buy and short[2] the stock; *portfolio strategies*, such as CRP, operate on the stock market as a whole and dictate how to allocate wealth among multiple stocks. We present several examples of common trading and portfolio strategies that can be universalized in our framework. We discuss our universalization framework in section 3. The proofs of our results are very general, and, as with previous universal portfolio results, we make no assumptions on the underlying distribution of the stock prices; our results are applicable for all sequences of stock returns and market conditions. The running times of universalization algorithms are, in general, exponential in the number of parameters used by the underlying investment strategy. Kalai and Vempala [14] presented an efficient implementation of the CRP algorithm that runs in time polynomial in the number of parameters. In section 4, we present general conditions on investment strategies under which the universalization algorithm can be efficiently implemented. We also give some investment strategies that satisfy these conditions. Section 5 concludes with directions for further research.

**2. Types of investment strategies.** Suppose we would like to distribute our wealth among $m$ stocks.[3] *Investment strategies* are general classes of rules that dictate how to invest capital. At time $t > 0$, a strategy $S$ takes as input an *environment vector* $\mathcal{E}_t$ and a *parameter vector* $\mathbf{w}$, and returns an *investment description* $S_t(\mathbf{w})$ specifying how to allocate our capital at time $t$. The environment vector $\mathcal{E}_t$ contains historic

[1]The average daily log-performance is the average of the logarithms of the factors by which our wealth changes on a daily basis. This notion is discussed further in section 3.1.

[2]A short position in a stock, discussed in section 2.1, allows us to earn a profit when the stock declines in value.

[3]We use the term “stocks” in order to keep our terminology consistent with previous work, but we actually mean a broader range of investment instruments, including both long and short positions in stocks.

market information, including stock price history, trading volumes, etc.; the parameter vector $\mathbf{w}$ is independent of $\mathcal{E}_t$ and specifies exactly how the strategy $S$ should operate; the investment description $S_t(\mathbf{w}) = (S_{t1}(\mathbf{w}), \ldots, S_{tm}(\mathbf{w}))$ is a vector specifying the proportion of wealth to put in each stock, where we put a fraction $S_{ti}(\mathbf{w})$ of our holdings in stock $i$, for $1 \leq i \leq m$. For example, CRP is an investment strategy; coupled with a portfolio vector $\mathbf{w}$, it tells us to "rebalance our portfolio on a daily basis according to $\mathbf{w}$"; its investment description, $\mathrm{CRP}_t(\mathbf{w}) = \mathbf{w}$, is independent of the market environment $\mathcal{E}_t$.

There are two general types of investment strategies that we focus upon in this paper. *Trading strategies* tell us whether we should take a *long* (bet that the stock price will rise) or a *short* (bet that the stock price will fall) position on a given stock. *Portfolio strategies* tell us how to distribute our wealth among various stocks. We should note here that these two classes do not exhaust all investment strategies; there exist strategies that take both long and short positions in several stocks (as in [21]). Trading strategies are denoted by $T$, and portfolio strategies are denoted by $P$. We use $S$ to denote either kind of strategy. For $k \geq 2$, let

$$\mathcal{W}_k = \left\{ \mathbf{w} = (w_1, \ldots, w_k) \in [0,1]^k \mid \sum_{i=1}^{k} w_i = 1 \right\}. \tag{2.1}$$

REMARK 1. *$\mathcal{W}_k$ is a $(k-1)$-dimensional simplex in $\mathbb{R}^k$. The investment strategies that we describe below are parameterized by vectors in $\mathcal{W}_k^\ell = \mathcal{W}_k \times \cdots \times \mathcal{W}_k$ ($\ell$ times) for some $k \geq 2$ and $\ell \geq 1$. We may write $\mathbf{w} \in \mathcal{W}_k^\ell$ in the form $\mathbf{w} = (\mathbf{w}_1, \ldots, \mathbf{w}_\ell)$, where $\mathbf{w}_\iota = (w_{\iota 1}, \ldots, w_{\iota k})$ for $1 \leq \iota \leq \ell$.*

**2.1. Trading strategies.** Suppose that our market contains a single stock. We have $m = 2$ potential investments: either a *long position* or a *short position* in the stock. To take a *long position*, we buy shares in hopes that the share price will rise. We *close a long position* by selling the shares. The money we use to buy the shares is our *investment in the long position*; the *value* of the investment is the money we get when we close the position. If we let $p_t$ denote the stock price at the beginning of day $t$, the value of our investment will change by a factor of $x_t = \frac{p_{t+1}}{p_t}$ from day $t$ to $t+1$.

To take a *short position*, we borrow shares from our broker and sell them on the market in hopes that the share price will fall. We *close a short position* by buying the shares back and returning them to our broker. As collateral for the borrowed shares, our broker has a *margin requirement*: a fraction $\alpha$ of the value of the borrowed shares must be deposited in a *margin account*. Should the price of the security rise sufficiently, the collateral in our margin account will not be enough, and the broker will issue a *margin call*, requiring us to deposit more collateral. The margin requirement is our *investment in the short position*; the *value* of the investment is the money we get when we close the position.

LEMMA 2.1. *Let the margin requirement for a short position be $\alpha \in (0,1]$. Suppose that a short position is opened on day t and that the price of the underlying stock changes by a factor of $x_t = \frac{p_{t+1}}{p_t} < 1 + \alpha$ during the day. Then the value of our investment in the short position changes by a factor of $x_t' = 1 + \frac{1-x_t}{\alpha}$ during the day.*

*Proof.* Suppose that we have \$$v$ to deposit in the zero-interest margin account. Using this as our investment in the short position, we can sell \$$v/\alpha$ worth of shares. Combining the proceeds of the stock sale with our margin account balance, we will have a total of $v + v/\alpha$ dollars. At the end of the day, it will cost $x_t v/\alpha$ dollars to buy the shares back, and we will be left with $v + \frac{v}{\alpha} - x_t \frac{v}{\alpha}$ dollars, which is positive since $x_t < 1 + \alpha$. Thus, our investment of \$$v$ in the short position has changed by a factor of $1 + \frac{1-x_t}{\alpha}$, as claimed. $\square$

Should the price of the underlying stock change by a factor greater than $1 + \alpha$, we will lose more money than we initially put in. We will assume that the margin requirement $\alpha$ is sufficiently large that the daily price change of the stock is always less than $1 + \alpha$.

REMARK 2. *This assumption can be eliminated by purchasing a* call option *on the stock with some strike price $p < (1 + \alpha)p_t$. Should the stock price get too high, the call allows us to purchase the stock back for \$$p$. Though its price detracts from the performance of our short trading strategy, the call protects us from potentially unlimited losses due to rising stock price.*

If a short position is held for several days, assume that it is *rebalanced* at the beginning of each day: either part of the short is closed (if $x_t > 1$) or additional shares are shorted (if $x_t < 1$) so that the collateral in the margin account is exactly an $\alpha$ fraction of the value of the shorted shares. This ensures that the value of a short position changes by a factor $x'_t = 1 + \frac{1-x_t}{\alpha}$ each day. Treating short positions in this way, they can simply be viewed as any other stock, so trading strategies are effectively investment strategies that decide between two potential investments: a long or a short position in a given stock. The investment description of a trading strategy $T$ is $T_t = (T_{t1}, T_{t2})$, where $T_{t1}$ and $T_{t2}$ are the fractions of wealth to put in a long and short position, respectively.

REMARK 3. *Let $D = T_{t1} - T_{t2}/\alpha$ be the* net long position *of the investment description. In practice, if $D > 0$, investors should put a $D$ fraction of their money in the long position and a $1 - D$ fraction in cash; if $D < 0$, investors should invest $D$ in the short position and $1 - D$ in cash; if $D = 0$, investors should avoid the stock completely and keep all their money in cash. From a practical standpoint, it is desirable for the trading strategy to be* decisive*, i.e., $|D| = 1$, so that our allocation of money to the stock is always fully invested in the stock (either as a long or a short position). We show in section 3 that investment strategies that are continuous in their parameter spaces are universalizable. Though decisive trading strategies $T$ are discontinuous, they can be approximated by continuous strategies whose investment descriptions converge almost everywhere to $T_t$ as $t \to \infty$ (see, for example,* (2.3) *below).*

We now describe some commonly used and researched trading strategies [4, 10, 18, 22] and show how they can be parameterized.

*MA[k]: Moving average cross-over with k-day memory.* In traditional applications [10] of this rule, we compare the current stock price with the moving average over, say, the previous 200 days: if the price is above the moving average, we take a long position; otherwise we take a short position. Some generalizations of this rule have been made, where we compare a fast moving average (over, for example, the past 5 to 20 days) with a slow moving average (over the past 50 to 200 days). We generalize this rule further. Given day $t \geq 0$, let $\mathbf{v}_t = (v_{t1}, \ldots, v_{tk})$ be the *price-history vector* over the previous $k$ days, where $v_{tj}$ is the stock price on day $t - j$. Assume that the stock prices have been normalized such that $0 < v_{tj} \leq 1$. Let $(\mathbf{w}_F, \mathbf{w}_S) \in \mathcal{W}_k^2$ (where $\mathcal{W}_k$ is defined in (2.1)) be the weights used to compute the fast moving and slow moving averages, so that these averages on day $t$ are given by $\mathbf{w}_F \cdot \mathbf{v}_t$ and $\mathbf{w}_S \cdot \mathbf{v}_t$, respectively. Since the prices have been normalized to the interval $(0, 1]$, $-1 \leq (\mathbf{w}_F - \mathbf{w}_S) \cdot \mathbf{v}_t \leq 1$, let $g : [-1, 1] \to [0, 1]$ be the *long/short allocation function.* The idea is that $g((\mathbf{w}_F - \mathbf{w}_S) \cdot \mathbf{v}_t)$ represents the proportion of wealth that we invest in a long position. The full investment description for the MA $=$ MA$[k]$ trading strategy is

$$\mathrm{MA}_t(\mathbf{w}_F, \mathbf{w}_S) = \big(g((\mathbf{w}_F - \mathbf{w}_S)\cdot \mathbf{v}_t),\ 1 - g((\mathbf{w}_F - \mathbf{w}_S)\cdot \mathbf{v}_t)\big).$$

Note that the dimension of the parameter space for MA$[k]$ is $2(k-1)$ since each of $\mathbf{w}_F$ and $\mathbf{w}_S$ are taken from $(k-1)$-dimensional spaces. Possible functions for $g$ include

$$g_s(x) = \begin{cases} 0 & \text{if } x < 0, \\ 1 & \text{otherwise,} \end{cases} \qquad \text{(step function)} \tag{2.2}$$

$$g_{(t)}(x) = \begin{cases} 0 & \text{if } x < -\frac{1}{t}, \\ \frac{t}{2}(x + \frac{1}{t}) & \text{if } -\frac{1}{t} \le x \le \frac{1}{t}, \\ 1 & \text{if } \frac{1}{t} < x, \end{cases} \qquad \text{(linear step approximation)} \tag{2.3}$$

and the line

$$g_\ell(x) = \frac{x+1}{2} \tag{2.4}$$

that intersects $g_s(x)$ at the extreme points $x = \pm 1$ of its domain. Note that $g_{(t)}(x)$ is parameterized by the day $t$ during which it is called and that it converges to $g_s(x)$ on $[-1,1] \setminus \{0\}$ as $t$ increases.

Remark 4. *The long/short allocation function used in traditional applications of this rule is the step function $g_s(\cdot)$. As we see in section 3, in order for an investment strategy to be universalizable, its allocation function must be continuous, necessitating the continuous approximation $g_{(t)}(\cdot)$. The linear approximation $g_\ell(\cdot)$ can be used with the results of section 4, to allow for efficient computation of the universalization algorithm.*

*SR$[k]$: Support and resistance breakout with k-day memory.* Discussed as early as the work of Wyckoff [22] in 1910, this strategy uses the idea that the stock price trades in a *range* bounded by *support* and *resistance* levels. Should the price fall below the support level, the idea is that it will continue to fall, and a short position should be taken in the stock. Similarly, should the price rise above the resistance level, the idea is that it will continue to rise, and a long position should be taken in the stock. If the stock price remains between the support and resistance levels, the idea is that it will continue to trade in this range in an unpredictable pattern, and the stock should be avoided. Support and resistance levels are defined quite arbitrarily in practice, usually as the minimum and maximum prices over the past $k$ days, where $k$ is usually taken to be 50, 150, or 200 [4]. To generalize this rule, given day $t \ge 0$, let $\underline{\mathbf{v}}_t = (\underline{v}_{t1}, \dots, \underline{v}_{tk})$ and $\overline{\mathbf{v}}_t = (\overline{v}_{t1}, \dots, \overline{v}_{tk})$ be the minimum and maximum price histories, where $\underline{v}_{tj}$ and $\overline{v}_{tj}$ are the minimum and maximum prices over the previous $j$ days, normalized so that they are in the range $(0,1]$. Let $\mathbf{w} \in \mathcal{W}_k$ be the weights for computing the support and resistance levels, so that these levels on day $t$ are given by $s_t = \mathbf{w} \cdot \underline{\mathbf{v}}_t$ and $r_t = \mathbf{w} \cdot \overline{\mathbf{v}}_t$, respectively.

Lemma 2.2. *The support level is bounded above by the resistance level: $s_t \le r_t$.*

*Proof.* This follows from the fact that for all $1 \le j \le k$, $\underline{v}_{tj} \le \overline{v}_{tj}$. $\square$

The long/short allocation function will be denoted by $h : \{(x,y) \in [-1,1]^2 \,|\, x \le y\} \to [0,1]$. Let $p_t$ be the current stock price (normalized to $(0,1]$ along with $\underline{\mathbf{v}}_t$ and $\overline{\mathbf{v}}_t$). The idea is that $h(p_t - r_t, p_t - s_t)$ tells us the proportion of wealth that we invest in a long position. The full investment description for the $\mathrm{SR} = \mathrm{SR}[k]$ trading strategy is

$$\mathrm{SR}_t(\mathbf{w}) = \big(h(p_t - r_t, p_t - s_t),\ 1 - h(p_t - r_t, p_t - s_t)\big).$$

The value of $h$ need only be defined on $\{(x,y) \in [-1,1]^2 \mid x \le y\}$ since, by Lemma 2.2, $s_t \le r_t$. A possible function for $h$ is

$$h_s(x,y) = \begin{cases} 0 & \text{if } x \le y \le 0, \\ \frac{1}{\alpha+1} & \text{if } x < 0 < y, \\ 1 & \text{if } y \ge x \ge 0, \end{cases} \qquad \text{(step function)} \tag{2.5}$$

where the investment allocation $\frac{1}{\alpha+1}$ long, $1 - \frac{1}{\alpha+1} = \frac{\alpha}{\alpha+1}$ short is equivalent to having no position in the stock, since the return from such an allocation is $\frac{x_t}{\alpha+1} + (1 + \frac{1-x_t}{\alpha})\frac{\alpha}{\alpha+1} = 1$. Other possibilities include a continuous function

$$h_{(t)}(x,y), \tag{2.6}$$

which approximates $h_s(x,y)$ with maximum slope at most $\frac{1}{t}$ (defined similarly to $g_{(t)}(x)$), or the plane

$$h_p(x,y) = \frac{(x+1)\alpha}{2(\alpha+1)} + \frac{y+1}{2(\alpha+1)} \tag{2.7}$$

that intersects $h_s(x,y)$ at the extreme points $(x,y) = (-1,-1)$, $(-1,1)$, and $(1,1)$ of its domain.

**2.2. Portfolio strategies.** Portfolio strategies are investment strategies that distribute wealth among $m$ stocks. The investment description of a portfolio strategy $P$ is $P_t = (P_{t1}, \ldots, P_{tm})$, where $0 \le P_{ti} \le 1$ and $\sum_{i=1}^{m} P_{ti} = 1$. We put a fraction $P_{ti}$ of our wealth in stock $i$ at time $t$.

*CRP: Constantly rebalanced portfolio* [5]. The parameter space for the CRP strategy is $\mathbb{W} = \mathcal{W}_m$. The investment description is $\mathrm{CRP}_t(\mathbf{w}) = \mathbf{w}$: at the beginning of each day, we invest a $w_i$ proportion of our wealth in stock $i$.

*CRP-S: Constantly rebalanced portfolio with side information.* Cover and Ordentlich [6] consider a generalization of CRP. Rather than rebalancing our holdings according to a single portfolio vector $\mathbf{w} \in \mathcal{W}_m$ every day, we have $k$ vectors $\mathbf{w}_1, \ldots, \mathbf{w}_k \in \mathcal{W}_m$ and a side information state $y_t \in \{1, \ldots, k\}$ that classifies each day $t$ into one of $k$ possible categories; on day $t$ we rebalance our holdings according to $\mathbf{w}_{y_t}$. By partitioning the time interval into $k$ subsequences corresponding to each of the $k$ side information states and running $k$ instances of the universalization algorithm (one instance for each state), Cover and Ordentlich show that the average daily return approaches that of the underlying strategy operating with $k$ optimal parameters, $\mathbf{w}_1^*, \ldots, \mathbf{w}_k^* \in \mathcal{W}_m$, where $\mathbf{w}_j^*$ is used on days $t$ when the side information state is $y_t = j$. We generalize this further by allowing portions of our wealth to be rebalanced according to several of the $\mathbf{w}_j$ every day. Suppose that the side information is encapsulated in some vector $\mathbf{v} \in \mathbb{R}^\ell$ for some $\ell$. This vector can contain information about specific stocks, such as historic performance and company fundamentals, or macroeconomic indicators such as inflation and unemployment. Let $\mathbf{f} = (f_1, \ldots, f_k) : \mathbb{R}^\ell \to [0,1]^k$ be some function satisfying

$\sum_{j=1}^{k} f_j(\mathbf{v}) = 1$ for all $\mathbf{v} \in \mathbb{R}^\ell$. The parameter space is $\mathcal{W}_m^k$; the investment description is $\text{CRP-S}_t(\mathbf{w}_1, \dots, \mathbf{w}_k) = \sum_{j=1}^{k} f_j(\mathbf{v}_t)\mathbf{w}_j$, where $\mathbf{v}_t$ is the indicator vector for day $t$. Under such a scheme, we have the flexibility of splitting our wealth among multiple sets of portfolios $\mathbf{w}_1, \dots, \mathbf{w}_k$ on any given day, rather than being forced to choose a single one. For example, assume that $\mathbf{v}$ is a $k$-dimensional vector, with each $v_i$ corresponding to portfolio $\mathbf{w}_i$. Define $\mathbf{f} : \mathbb{R}^k \to [0,1]^k$ by $f_i(\mathbf{v}_t) = \frac{v_{ti}}{\sum_{\iota=1}^{k} v_{t\iota}}$, so that our allocation is biased towards portfolios corresponding to higher indicators while still maintaining a position in the others.

*IA*[$k$]*: $k$-way indicator aggregation.* For each day $t \geq 0$, suppose that each stock $i$ has a set of $k$ indicators $\mathbf{v}_{ti} = (v_{ti1}, \dots, v_{tik})$, where each $v_{tij} \in (0,1]$ and, for $1 \leq j \leq k$, $v_{t1j}, \dots, v_{tmj}$ have been normalized such that there is at least one $i$ such that $v_{tij} = 1$. Examples of possible indicators include historic stock performance and trading volumes, and company fundamentals. Our goal is to aggregate the indicators for each stock to get a measure of the stock's attractiveness and put a greater proportion of our wealth in stocks that are more attractive. We will aggregate the indicators by taking their weighted average, where the weights will be determined by the parameters. The parameter space is $\mathbb{W} = \mathcal{W}_k$, and the investment description is

$$\text{IA}_t(\mathbf{w}) = \left( \frac{\mathbf{w} \cdot \mathbf{v}_{t1}}{\sum_{i=1}^{m} \mathbf{w} \cdot \mathbf{v}_{ti}}, \dots, \frac{\mathbf{w} \cdot \mathbf{v}_{tm}}{\sum_{i=1}^{m} \mathbf{w} \cdot \mathbf{v}_{ti}} \right).$$

## 3. Universalization of investment strategies.

**3.1. Universalization defined.** In a typical stock market, wealth grows geometrically. On day $t \geq 0$, let $\mathbf{x}_t$ be the *return vector* for day $t$, the vector of factors by which stock prices change on day $t$. The return vector corresponding to a trading strategy on a single stock is $(x_t, 1 + \frac{1-x_t}{\alpha})$, where $x_t$ is the factor by which the price of the stock changes and $1 + \frac{1-x_t}{\alpha}$ is the factor by which our investment in a short position changes, as described in Lemma 2.1; the return vector corresponding to a portfolio strategy is $(x_{t1}, \dots, x_{tm})$, where $x_{ti}$ is the factor by which the price of stock $i$ changes, where $1 \leq i \leq m$. Henceforth, we do not make a distinction between return vectors corresponding to trading and portfolio strategies; we assume that $\mathbf{x}_t$ is appropriately defined to correspond to the investment strategy in question. For an investment strategy $S$ with parameter vector $\mathbf{w}$, the *return of* $S(\mathbf{w})$ during the $t$th day—the factor by which our wealth changes on the $t$th day when invested according to $S(\mathbf{w})$—is $S_t(\mathbf{w}) \cdot \mathbf{x}_t = \sum_{i=1}^{m} S_{ti}(\mathbf{w}) \cdot x_{ti}$. (Recall that $S_t(\mathbf{w})$ is the investment description of $S(\mathbf{w})$ for day $t$, which is a vector specifying the proportion of wealth to put in each stock.) Given time $n > 0$, let $\mathcal{R}_n(S(\mathbf{w})) = \prod_{t=0}^{n-1} S_t(\mathbf{w}) \cdot \mathbf{x}_t$ be the *cumulative return of* $S(\mathbf{w})$ *up to time* $n$; we may write $\mathcal{R}_n(\mathbf{w})$ in place of $\mathcal{R}_n(S(\mathbf{w}))$ if $S$ is obvious from context. We analyze the performance of $S$ in terms of the *normalized log-return* $\mathcal{L}_n(\mathbf{w}) = \mathcal{L}_n(S(\mathbf{w})) = \frac{1}{n} \log \mathcal{R}_n(\mathbf{w})$ of the wealth achieved.

For investment strategy $S$, let $\mathbf{w}_n^* = \arg\max_{\mathbf{w} \in \mathbb{R}^*} \mathcal{R}_n(S(\mathbf{w}))$ be the parameters that maximize the return of $S$ up to day $n$.[4] An investment strategy $U$ *universalizes* (or *is universal for*) $S$ if[5]

$$\mathcal{L}_n(U) = \mathcal{L}_n(S(\mathbf{w}_n^*)) - \text{o}(1)$$

[4]As mentioned above, $\mathbf{w}_n^*$ can be computed only with hindsight.

[5]Unlike previously discussed investment strategies, the behavior of $U$ is fully defined without an additional parameter vector $\mathbf{w}$.

for all environment vectors $\mathcal{E}_n$. That is, $U$ is universal for $S$ if the average daily log-return of $U$ approaches the optimal average daily log-return of $S$ as the length $n$ of the time horizon grows, regardless of stock price sequences.

**3.2. General techniques for universalization.** Given an investment strategy $S$, let $\mathbb{W}$ be the parameter space for $S$ and let $\mu$ be the uniform measure over $\mathbb{W}$. Our universalization algorithm for $S$, $\mathcal{U}(S)$, is a generalization of Cover's original result [5]; we note that a similar algorithm appeared in [20] under the name "Aggregating Algorithm." The investment description $\mathcal{U}_t(S)$ for the universalization of $S$ on day $t > 0$ is a weighted average of $S_t(\mathbf{w})$ over $\mathbf{w} \in \mathbb{W}$, with greater weight given to parameters $\mathbf{w}$ that have performed better in the past (i.e., $\mathcal{R}_t(\mathbf{w})$ is larger). Formally, the investment description is

$$\mathcal{U}_t(S) = \frac{\int_{\mathbb{W}} S_t(\mathbf{w})\mathcal{R}_t(\mathbf{w})d\mu(\mathbf{w})}{\int_{\mathbb{W}} \mathcal{R}_t(\mathbf{w})d\mu(\mathbf{w})} = \frac{\int_{\mathbb{W}} S_t(\mathbf{w})\mathcal{R}_t(S(\mathbf{w}))d\mu(\mathbf{w})}{\int_{\mathbb{W}} \mathcal{R}_t(S(\mathbf{w}))d\mu(\mathbf{w})}, \tag{3.1}$$

where we take $\mathcal{R}_0(\mathbf{w}) = 1$ for all $\mathbf{w} \in \mathbb{W}$.[6] Equivalently, the above integral might be interpreted as "splitting our money" equally among all the different strategies and "letting it sit." In the following lemma, we will prove that this strategy has the same expected gain as picking one strategy at random.

REMARK 5. *The definition of universalization can be expanded to include measures other than $\mu$, but we consider only $\mu$ in our results.*

LEMMA 3.1 (see [3, 6]). *The cumulative $n$-day return of $\mathcal{U}(S)$ is*

$$\mathcal{R}_n(\mathcal{U}(S)) = \int_{\mathbb{W}} \mathcal{R}_n(\mathbf{w})d\mu(\mathbf{w}) = \mathbb{E}\big(\mathcal{R}_n(\mathbf{w})\big),$$

*which is the $\mu$-weighted average of the cumulative returns of the investment strategies $\{S(\mathbf{w}) \mid \mathbf{w} \in \mathbb{W}\}$.*

*Proof.* The return of $\mathcal{U}(S)$ on day $t$ is $\mathcal{U}_t(S) \cdot \mathbf{x}_t$, where $\mathbf{x}_t$ is the return vector for day $t$. The cumulative $n$-day return of $\mathcal{U}(S)$ is

$$\begin{aligned}\mathcal{R}_n(\mathcal{U}(S)) &= \prod_{t=0}^{n-1} \mathcal{U}_t(S) \cdot \mathbf{x}_t = \prod_{t=0}^{n-1} \frac{\int_{\mathbb{W}} S_t(\mathbf{w})\mathcal{R}_t(\mathbf{w})d\mu(\mathbf{w})}{\int_{\mathbb{W}} \mathcal{R}_t(\mathbf{w})d\mu(\mathbf{w})} \cdot \mathbf{x}_t \\ &= \prod_{t=0}^{n-1} \frac{\int_{\mathbb{W}} (S_t(\mathbf{w}) \cdot \mathbf{x}_t)\mathcal{R}_t(\mathbf{w})d\mu(\mathbf{w})}{\int_{\mathbb{W}} \mathcal{R}_t(\mathbf{w})d\mu(\mathbf{w})} = \prod_{t=0}^{n-1} \frac{\int_{\mathbb{W}} \mathcal{R}_{t+1}(\mathbf{w})d\mu(\mathbf{w})}{\int_{\mathbb{W}} \mathcal{R}_t(\mathbf{w})d\mu(\mathbf{w})}.\end{aligned}$$

The result follows from the fact that this product telescopes. □

Rather than directly universalizing a given investment strategy $S$, we instead focus on a modified version of $S$ that puts a nonzero fraction of wealth into each of the $m$ stocks. Define the investment strategy $\bar{S}$ by

$$\bar{S}_t(\mathbf{w}) = \left(1 - \frac{\varepsilon}{2(t+1)^2}\right) S_t(\mathbf{w}) + \frac{\varepsilon}{2m(t+1)^2}$$

for $t \geq 0$ and some fixed $0 < \varepsilon < 1$. Rather than universalizing $S$, we instead universalize $\bar{S}$. Lemma 3.2 tells us that we do not lose much by doing this.

LEMMA 3.2. *For all $n \geq 0$, (1) $\mathcal{R}_n(\mathcal{U}(\bar{S})) \geq (1-\varepsilon)\mathcal{R}_n(\mathcal{U}(S))$ and (2) $\mathcal{L}_n(\mathcal{U}(\bar{S})) = \mathcal{L}_n(\mathcal{U}(S)) - \frac{o(n)}{n}$. (3) If $\mathcal{U}(S)$ is a universalization of $S$, then $\mathcal{U}(\bar{S})$ is a universalization of $S$ as well.*

[6] Cover's algorithm is a special case of this, replacing $S_t(\mathbf{w})$ with $\mathbf{w}$.

*Proof.* Statements (2) and (3) follow directly from (1). Statement (1) follows from the fact that for all $\mathbf{w} \in \mathbb{W}$, $\mathcal{R}_n(\bar{S}(\mathbf{w})) = \prod_{t=0}^{n-1} \bar{S}_t(\mathbf{w}) \cdot \mathbf{x}_t \geq \prod_{t=0}^{n-1}(1 - \frac{\varepsilon}{2(t+1)^2})S_t(\mathbf{w}) \cdot \mathbf{x}_t \geq (1 - \sum_{t=0}^{n-1} \frac{\varepsilon}{2(t+1)^2})\mathcal{R}_n(S(\mathbf{w})) \geq (1-\varepsilon)\mathcal{R}_n(S(\mathbf{w}))$. ◻

REMARK 6. *Henceforth, we assume that suitable modifications have been made to $S$ to ensure that $S_{ti}(\mathbf{w}) \geq \frac{\varepsilon}{2m(t+1)^2}$ for all $1 \leq i \leq m$ and $t \geq 0$.*

THEOREM 3.3. *Given an investment strategy $S$, let $\mathbb{W} = \mathcal{W}_k^\ell$ (for some $k \geq 2$ and $\ell \geq 1$) be its parameter space. For $1 \leq i \leq m$, $1 \leq \iota \leq \ell$, and $1 \leq j \leq k$, assume that there is a constant $c$ such that $\left|\frac{\partial S_{ti}(\mathbf{w})}{\partial w_{\iota j}}\right| \leq c(t+1)$ for all $\mathbf{w} \in \mathbb{W}$. Then $\mathcal{U}(S)$ is a universalization of $S$.*

To prove Theorem 3.3, we first prove some preliminary results; the proof follows the same general strategy as in [5, 6].

LEMMA 3.4. *For nonnegative vector $\mathbf{a}$ and strictly positive vectors $\mathbf{b}$ and $\mathbf{x}$,*

$$\min_i \frac{a_i}{b_i} \leq \frac{\mathbf{a} \cdot \mathbf{x}}{\mathbf{b} \cdot \mathbf{x}} \leq \max_i \frac{a_i}{b_i}.$$

*Proof.* Assume that the components of $\mathbf{a}$ and $\mathbf{b}$ are strictly positive. Otherwise, the lemma holds trivially. Let $i_{\max} = \arg\max_i \frac{a_i}{b_i}$ and $i_{\min} = \arg\min_i \frac{a_i}{b_i}$, so that

$$\frac{a_i}{b_i} \leq \frac{a_{i_{\max}}}{b_{i_{\max}}} \Leftrightarrow \frac{a_i}{a_{i_{\max}}} \leq \frac{b_i}{b_{i_{\max}}} \quad \text{and} \quad \frac{a_i}{b_i} \geq \frac{a_{i_{\min}}}{b_{i_{\min}}} \Leftrightarrow \frac{a_i}{a_{i_{\min}}} \geq \frac{b_i}{b_{i_{\min}}}.$$

Then

$$\frac{a_{i_{\min}}(x_{i_{\min}} + \sum_{i \neq i_{\min}} \frac{a_i}{a_{i_{\min}}} x_i)}{b_{i_{\min}}(x_{i_{\min}} + \sum_{i \neq i_{\min}} \frac{b_i}{b_{i_{\min}}} x_i)} = \frac{\mathbf{a} \cdot \mathbf{x}}{\mathbf{b} \cdot \mathbf{x}} = \frac{a_{i_{\max}}(x_{i_{\max}} + \sum_{i \neq i_{\max}} \frac{a_i}{a_{i_{\max}}} x_i)}{b_{i_{\max}}(x_{i_{\max}} + \sum_{i \neq i_{\max}} \frac{b_i}{b_{i_{\max}}} x_i)}.$$

Therefore,

$$\frac{a_{i_{\min}}}{b_{i_{\min}}} \leq \frac{\mathbf{a} \cdot \mathbf{x}}{\mathbf{b} \cdot \mathbf{x}} \leq \frac{a_{i_{\max}}}{b_{i_{\max}}}. \quad ◻$$

Our next two results are related to the $(k-1)$-dimensional volumes of some subsets of $\mathbb{R}^k$.

LEMMA 3.5. *The $(k-1)$-dimensional volume of the simplex $\mathcal{W}_k = \{\mathbf{w} \in [0,1]^k \mid \sum_{i=1}^k w_i = 1\}$, defined in (2.1), is $\frac{\sqrt{k}}{(k-1)!}$.*

*Proof.* By induction on $k$, it can be shown that the $k$-dimensional volume of the solid $W_k(s) = \{\mathbf{w} \mid \sum_{i=1}^k w_i \leq s\}$ is $\frac{s^k}{k!}$. Written in terms of the length $r$ of the line segment passing between the origin and $(\frac{s}{k}, \ldots, \frac{s}{k}) \in \mathbb{R}^k$, the volume is $\frac{1}{k!} r^k k^{\frac{k}{2}}$ since $s = r\sqrt{k}$. Upon differentiation with respect to $r$, $\frac{1}{(k-1)!} r^{k-1} k^{\frac{k}{2}} = \frac{1}{(k-1)!}\sqrt{k} s^{k-1}$, we arrive at the $(k-1)$-dimensional volume of the simplex $\mathcal{W}_k(s) = \{\mathbf{w} \mid \sum_{i=1}^k w_i = s\}$. Setting $s = 1$ yields the desired result. ◻

LEMMA 3.6. *The $(k-1)$-dimensional volume of a $(k-1)$-dimensional ball of radius $\rho$ embedded in $\mathcal{W}_k$ is $\frac{\pi^{\frac{k-1}{2}} \rho^{k-1}}{\Gamma(\frac{k-1}{2}+1)}$, where*

$$\Gamma(\ell) = (\ell - 1)! \quad \textit{and} \quad \Gamma\left(\ell + \frac{1}{2}\right) = \left(\ell - \frac{1}{2}\right)\left(\ell - \frac{3}{2}\right) \cdots \left(\frac{1}{2}\right)\sqrt{\pi}.$$

*Proof.* This result is proven in Folland [8, Corollary 2.56]. ◻

*Proof of Theorem* 3.3. From Lemma 3.1, the return of $\mathcal{U}(S)$ is the average of the cumulative returns of the investment strategies $\{S(\mathbf{w}) \mid \mathbf{w} \in \mathbb{W}\}$. Let $\mathbf{w}^* = \arg\max_{\mathbf{w}\in\mathbb{W}} \mathcal{R}_n(S(\mathbf{w}))$ be the parameters that maximize the return of $S$. We show that there is a set $B$ of nonzero volume around $\mathbf{w}^*$ such that for $\mathbf{w} \in B$ the return $\mathcal{R}_n(\mathbf{w})$ is close to the optimal return $\mathcal{R}_n(\mathbf{w}^*)$. We then show that the contribution of $B$ to the average return is sufficiently large to ensure universalizability. We begin by bounding the magnitude of the gradient vector $\nabla\mathcal{R}_n(\mathbf{w})$. From Remark 6 and our assumption in the statement of the theorem, for all $\mathbf{w}$, $t$, $i$, $\iota$, and $j$,

$$\frac{\left|\frac{\partial S_{ti}(\mathbf{w})}{\partial w_{\iota j}}\right|}{S_{ti}(\mathbf{w})} \le c'm(t+1)^3,$$

where $c' = \frac{2c}{\varepsilon}$. Using this fact and Lemma 3.4, the partial derivative of the return function $\mathcal{R}_n(\mathbf{w}) = \mathcal{R}_n(S(\mathbf{w})) = \prod_{t=0}^{n-1} r_t(S(\mathbf{w}))$ with respect to parameter $w_{\iota j}$ is

$$\begin{aligned}\left|\frac{\partial\mathcal{R}_n(\mathbf{w})}{\partial w_{\iota j}}\right| &\le \mathcal{R}_n(\mathbf{w}) \sum_{t=0}^{n-1} \frac{\left|\frac{\partial(S_t(\mathbf{w})\cdot\mathbf{x}_t)}{\partial w_{\iota j}}\right|}{S_t(\mathbf{w})\cdot\mathbf{x}_t} \le \mathcal{R}_n(\mathbf{w}) \sum_{t=0}^{n-1} \frac{\sum_{i=1}^m \left|\frac{\partial S_{ti}(\mathbf{w})}{\partial w_{\iota j}}\right| \cdot x_{ti}}{\sum_{i=1}^m S_{ti}(\mathbf{w})\cdot x_{ti}} \\ &\le \mathcal{R}_n(\mathbf{w}) \sum_{t=0}^{n-1} c'm(t+1)^3 \le c'\mathcal{R}_n(\mathbf{w})mn^4\end{aligned}$$

and

$$|\nabla\mathcal{R}_n(\mathbf{w})| \le c'\mathcal{R}_n(\mathbf{w})mn^4\sqrt{k\ell}. \tag{3.2}$$

We would like to take our set $B$ to be some $d$-dimensional ball around $\mathbf{w}^*$; unfortunately, if $\mathbf{w}^*$ is on (or close to) an edge of $\mathbb{W}$, the reasoning introduced at the beginning of this proof is not valid. We instead perturb $\mathbf{w}^*$ to a point $\tilde{\mathbf{w}}$ that is at least

$$\rho = \frac{\gamma}{c'mn^4k^2\ell}$$

away from all edges, where $0 < \gamma < 1$ is a constant, and such that $\mathcal{R}_n(\tilde{\mathbf{w}})$ is close to $\mathcal{R}_n(\mathbf{w}^*)$. To illustrate the perturbation, let $\mathbf{w}^* = (\mathbf{w}_1^*, \ldots, \mathbf{w}_\ell^*)$, where $\mathbf{w}_\iota^* = (w_{\iota 1}^*, \ldots, w_{\iota k}^*)$ and $w_{\iota k}^* = 1 - \sum_{i=1}^{k-1} w_{\iota i}^*$ for $1 \le \iota \le \ell$. We perturb each $\mathbf{w}_\iota^*$ in the same way. Let $\tilde{\mathbf{w}}_\iota^0 = \mathbf{w}_\iota^*$. For $1 \le j \le k$, given $\tilde{\mathbf{w}}_\iota^{j-1}$, define $\tilde{\mathbf{w}}_\iota^j$ as follows. Let $j_{\max}$ be the index of the maximum coordinate of $\tilde{\mathbf{w}}_\iota^{j-1}$. If $0 \le \tilde{w}_{\iota j}^j < \rho$, define $\tilde{w}_{\iota j}^j = \tilde{w}_{\iota j}^{j-1} + \rho$, $\tilde{w}_{\iota j_{\max}}^j = \tilde{w}_{\iota j_{\max}}^{j-1} - \rho$ and leave all other coordinates unchanged. Otherwise, let $\tilde{\mathbf{w}}_{j_0}^j = \tilde{\mathbf{w}}_{j_0}^{j-1}$. The final perturbation is $\tilde{\mathbf{w}} = (\tilde{\mathbf{w}}_1, \ldots, \tilde{\mathbf{w}}_\ell)$, where $\tilde{\mathbf{w}}_\iota = \tilde{\mathbf{w}}_\iota^k$. By construction, $\tilde{\mathbf{w}} \in \mathbb{W}$, $\tilde{\mathbf{w}}$ is at least $\rho$ away from the edges of $\mathbb{W}$ and $|w_{\iota j}^* - \tilde{w}_{\iota j}| \le k\rho$ for all $\iota$ and $j$. We bound $\frac{\mathcal{R}_n(\mathbf{w}^*)}{\mathcal{R}_n(\tilde{\mathbf{w}})}$ by the multivariate mean value theorem and the Cauchy–Schwarz inequality:

$$\begin{aligned}\mathcal{R}_n(\tilde{\mathbf{w}}) &= \mathcal{R}_n(\mathbf{w}^*) + \mathcal{R}_n(\tilde{\mathbf{w}}) - \mathcal{R}_n(\mathbf{w}^*) \\ &\ge \mathcal{R}_n(\mathbf{w}^*) - |\nabla\mathcal{R}_n(\mathbf{w}')\cdot(\tilde{\mathbf{w}} - \mathbf{w}^*)| \qquad (\text{for some } \mathbf{w}' \text{ between } \tilde{\mathbf{w}} \text{ and } \mathbf{w}^*) \\ &\ge \mathcal{R}_n(\mathbf{w}^*) - |\nabla\mathcal{R}_n(\mathbf{w}')|\cdot|\tilde{\mathbf{w}} - \mathbf{w}^*| \ge \mathcal{R}_n(\mathbf{w}^*) - c'\mathcal{R}_n(\mathbf{w}')mn^4\sqrt{k\ell}\cdot k\rho\sqrt{k\ell} \\ &\ge \mathcal{R}_n(\mathbf{w}^*) - c'\mathcal{R}_n(\mathbf{w}^*)mn^4\sqrt{k\ell}\cdot k\rho\sqrt{k\ell} \ge \mathcal{R}_n(\mathbf{w}^*)(1-\gamma).\end{aligned}$$

For $0 \le \iota \le \ell$ let $C_\iota = \{\mathbf{w}_\iota \in \mathbb{R}^k \mid |\tilde{\mathbf{w}}_\iota - \mathbf{w}_\iota| \le \rho\}$. From the construction of $\tilde{\mathbf{w}}$, $B_\iota = C_\iota \cap \mathcal{W}_k$ is a $(k-1)$-dimensional ball of radius $\rho$. Let $\tilde{\mathbf{w}}^*_\iota = \arg\max_{\mathbf{w}\in B_\iota} \mathcal{R}_n(\mathbf{w})$, and let $\tilde{\mathbf{w}}^* = (\tilde{\mathbf{w}}^*_1, \dots, \tilde{\mathbf{w}}^*_\ell)$ be the profit-maximizing parameters in $B = B_1 \times \cdots \times B_\ell$. For $\mathbf{w} \in B$,

$$\begin{aligned}
\mathcal{R}_n(\mathbf{w}) &= \mathcal{R}_n(\tilde{\mathbf{w}}^*) + \mathcal{R}_n(\mathbf{w}) - \mathcal{R}_n(\tilde{\mathbf{w}}^*) \\
&\ge \mathcal{R}_n(\tilde{\mathbf{w}}^*) - |\nabla \mathcal{R}_n(\mathbf{w}')| \cdot |\tilde{\mathbf{w}}^* - \mathbf{w}| \qquad (\text{for some } \mathbf{w}' \text{ between } \tilde{\mathbf{w}}^* \text{ and } \mathbf{w}) \\
&\ge \mathcal{R}_n(\tilde{\mathbf{w}}^*) - c'\mathcal{R}_n(\tilde{\mathbf{w}}^*) m n^4 \sqrt{k\ell} \cdot 2\rho\sqrt{\ell} \ge \mathcal{R}_n(\tilde{\mathbf{w}}^*)(1-\gamma) \\
&\ge \mathcal{R}_n(\mathbf{w}^*)(1-2\gamma).
\end{aligned}$$

By Lemma 3.1,

$$\begin{aligned}
\mathcal{R}_n(\mathcal{U}(S)) &= \int_{\mathbb{W}} \mathcal{R}_n(S(\mathbf{w})) d\mu(\mathbf{w}) \ge \int_B \mathcal{R}_n(\mathbf{w}) d\mu(\mathbf{w}) \ge (1-2\gamma)\mathcal{R}_n(\mathbf{w}^*) \int_B d\mu(\mathbf{w}) \\
&\ge (1-2\gamma)\mathcal{R}_n(\mathbf{w}^*) \frac{\int_B d\mathbf{w}}{\int_{\mathbb{W}} d\mathbf{w}} \\
&= (1-2\gamma)\mathcal{R}_n(\mathbf{w}^*) \left( \frac{\pi^{\frac{k-1}{2}} \rho^{k-1}}{\Gamma(\frac{k-1}{2}+1)} \cdot \frac{(k-1)!}{\sqrt{k}} \right)^\ell \qquad (\text{from Lemmas 3.5 and 3.6}) \\
&= \mathcal{R}_n(\mathbf{w}^*) \Lambda(\gamma, m, k, \ell) n^{-4k\ell},
\end{aligned}$$

where $\Lambda$ is some constant depending on $\gamma$, $m$, $k$, and $\ell$. Therefore,

$$\mathcal{L}_n(\mathbf{w}^*) - \mathcal{L}_n(\mathcal{U}(S)) \le \frac{\log \Lambda(\gamma, m, k, \ell)}{n} + 4k\ell \frac{\log n}{n} = \frac{\mathrm{o}(n)}{n}, \tag{3.3}$$

as claimed. □

REMARK 7. *The techniques used in the proof of Theorem* 3.3 *can be generalized to other investment strategies with bounded parameter spaces* $\mathbb{W}$ *that are not necessarily of the form* $\mathcal{W}_k^\ell$.

**3.3. Increasing the number of parameters with time.** The reader may notice from the proof of Theorem 3.3 that an investment strategy $S$ may be universalizable even if the dimensions of its parameter space $\mathbb{W}$ grow with time. In fact, even if the dimension of the parameter space (the coefficient of $\frac{\log n}{n}$ in (3.3)) is $\mathcal{O}(\frac{n}{\phi(n)\log n})$, where $\phi(n)$ is a monotone increasing function, the strategy is still universalizable. This introduces an interesting possibility for investment strategies whose parameter spaces grow with time as more information becomes available. As a simple example, consider *dynamic universalization*, which allows us to track a higher-return benchmark than basic universalization. Partition the time interval $\mathcal{I} = [0, n)$ into $\psi = \mathcal{O}(\frac{n}{\phi(n)\log n})$ subintervals $\mathcal{I}_1, \dots, \mathcal{I}_\psi$, and let $\mathbf{w}^*_{\mathcal{I}_j}$ be the parameters that optimize the return during $\mathcal{I}_j$. In $\mathcal{I}_1$, we run the universalization algorithm given by (3.1) over the basic parameter space $\mathbb{W}$ of $S$. In $\mathcal{I}_2$, we run the algorithm over $\mathbb{W} \times \mathbb{W}$; to compute the investment description for a day $t \in \mathcal{I}_2$ using (3.1), we compute the return $\mathcal{R}_t(\mathbf{w}_1, \mathbf{w}_2)$ as the product of the returns we would have earned in $\mathcal{I}_1$ using $\mathbf{w}_1$ and what we would have earned up to day $t$ in $\mathcal{I}_2$ using $\mathbf{w}_2$. We proceed similarly in intervals $\mathcal{I}_3$ through $\mathcal{I}_\psi$. This will allow us to track the strategy that uses the optimal parameters $\mathbf{w}^*_{\mathcal{I}_j}$ corresponding to each $\mathcal{I}_j$. Such a strategy is useful in environments where optimal investment styles (and the optimal investment strategy parameters that go with them) change with time. Finally, we note that similar ideas appear in the area of "tracking the best expert" in the theory of prediction with expert advice; we refer the reader to [12, 19] for more details.

**3.4. Applications to trading strategies.** By proving an upper bound on $\left|\frac{\partial T_{ti}(\mathbf{w})}{\partial w_j}\right|$ for our trading strategies $T$, we show that they are universalizable.

COROLLARY 3.7. *The moving average cross-over trading strategy, MA*$[k]$*, is universalizable for the long/short allocation functions* $g_{(t)}(x)$ *and* $g_\ell(x)$ *defined in* (2.3) *and* (2.4)*, respectively.*

*Proof.* The parameters for MA$[k]$ are of the form $\mathbf{w}_F = (w_{F1}, \dots, w_{F(k-1)}, 1 - w_{F1} - \cdots - w_{F(k-1)})$ and $\mathbf{w}_S = (w_{S1}, \dots, w_{S(k-1)}, 1 - w_{S1} - \cdots - w_{S(k-1)})$. Using the long/short allocation function $g_{(t)}(x)$ defined in (2.3), the partial derivative of the investment description with respect to a parameter $w_{Fj}$ (or similarly $w_{Sj}$) is

$$\left|\frac{\partial \mathrm{MA}_{ti}(\mathbf{w}_F, \mathbf{w}_S)}{\partial w_{Fj}}\right| = \left|\frac{\partial g((\mathbf{w}_F - \mathbf{w}_S)\cdot \mathbf{v}_t)}{\partial w_{Fj}}\right| \le \frac{t}{2}\cdot(v_{tj} - v_{tk}) \le \frac{t}{2},$$

where $1 \le j < k$ and $i \in \{1, 2\}$. Similarly, we can show that using the long/short allocation function $g_\ell(x)$ defined in (2.4), $\left|\frac{\partial \mathrm{MA}_{ti}(\mathbf{w}_F, \mathbf{w}_S)}{\partial w_{Fj}}\right| \le \frac{1}{2}$. ∎

COROLLARY 3.8. *The support and resistance breakout trading strategy, SR*$[k]$*, is universalizable for the long/short allocation functions* $h_{(t)}(x, y)$ *and* $h_p(x, y)$ *defined in* (2.6) *and* (2.7)*, respectively.*

*Proof.* We arrive at the result by differentiating the long/short allocation functions $h_{(t)}(x, y)$ and $h_p(x, y)$ with respect to an arbitrary parameter $w_j$ and showing that the partial derivative is $\mathcal{O}(t)$, as in the proof of Corollary 3.7. ∎

**3.5. Applications to portfolio strategies.**

COROLLARY 3.9. *The constantly rebalanced portfolio, CRP, and CRP with side information, CRP-S, portfolio strategies are universalizable.*

*Proof.* The partial derivatives of CRP$_{ti}$ and CRP-S$_{ti}$ with respect to an arbitrary parameter $w_j$ are at most 1. ∎

COROLLARY 3.10. *The* $k$*-way indicator aggregation portfolio strategy, IA*$[k]$*, is universalizable.*

*Proof.* First, we show that $\sum_{\ell=1}^m \mathbf{w}\cdot \mathbf{v}_{t\ell} \ge \frac{1}{k}$ for all $t$. Since $\sum_{j=1}^k w_j = 1$, there exists $j_0$ such that $w_{j_0} \ge \frac{1}{k}$. Then $\sum_{\ell=1}^m \mathbf{w}\cdot\mathbf{v}_{t\ell} \ge \sum_{\ell=1}^m w_{j_0}\cdot v_{t\ell j_0} \ge \frac{1}{k}\sum_{\ell=1}^m v_{t\ell j_0} \ge \frac{1}{k}$, since the $\{v_{t\ell j_0}\}_{1\le \ell\le m}$ have been normalized such that there is at least one $\ell_0$ such that $v_{t\ell_0 j_0} = 1$.

Now, let $S = \mathrm{IA}[k]$. By Theorem 3.3, we need only show that $\frac{\partial S_{ti}(\mathbf{w})}{\partial w_j} = \mathcal{O}(t)$ for $1 \le j \le k-1$. For $t \ge 0$ and $1 \le i \le m$ recall that $S_{ti}(\mathbf{w}) = \frac{\mathbf{w}\cdot\mathbf{v}_{ti}}{\sum_{\ell=1}^m \mathbf{w}\cdot\mathbf{v}_{t\ell}}$. Then, for $1 \le j \le k-1$, since $\mathbf{w} = (w_1, \dots, w_{k-1}, 1 - (w_1 + \cdots + w_{k-1}))$,

$$\begin{aligned}\frac{\partial S_{ti}(\mathbf{w})}{\partial w_j} &= \frac{v_{tij} - v_{tik}}{\sum_{\ell=1}^m \mathbf{w}\cdot\mathbf{v}_{t\ell}} - \frac{\mathbf{w}\cdot\mathbf{v}_{ti}}{(\sum_{\ell=1}^m \mathbf{w}\cdot\mathbf{v}_{t\ell})^2}\cdot\sum_{\ell=1}^m (v_{t\ell j} - v_{t\ell k}) \\ &\le \frac{1}{\sum_{\ell=1}^m \mathbf{w}\cdot\mathbf{v}_{t\ell}} + \frac{m}{(\sum_{\ell=1}^m \mathbf{w}\cdot\mathbf{v}_{t\ell})^2} \le k + mk^2,\end{aligned}$$

as we wanted to show. ∎

## 4. Fast computation of universal investment strategies.

**4.1. Approximation by sampling.** The running time of the universalization algorithm depends on the time needed to compute the integral in (3.1). A straightforward evaluation takes time exponential in the number of parameters. Following Kalai and Vempala [14], we propose to approximate this integral by sampling the parameters according to a biased distribution, giving greater weight to better performing parameters. Define the measure $\zeta_t$ on $\mathbb{W}$ by

$$d\zeta_t(\mathbf{w}) = \frac{\mathcal{R}_t(S(\mathbf{w}))}{\int_{\mathbb{W}} \mathcal{R}_t(S(\mathbf{w}))d\mu(\mathbf{w})} d\mu(\mathbf{w}).$$

LEMMA 4.1 (see [14]). *The investment description $\mathcal{U}_t(S)$ for universalization is the average of $S_t(\mathbf{w})$ with respect to the $\zeta_t$ measure.*

*Proof.* The average of $S_t(\mathbf{w})$ with respect to $\zeta_t$ is

$$\begin{aligned}\mathbb{E}_{\mathbf{w}\in(\mathbb{W},\zeta_t)}(S_t(\mathbf{w})) &= \int_{\mathbb{W}} S_t(\mathbf{w})d\zeta_t(\mathbf{w}) \\ &= \int_{\mathbb{W}} S_t(\mathbf{w}) \frac{\mathcal{R}_t(S(\mathbf{w}))}{\int_{\mathbb{W}} \mathcal{R}_t(S(\mathbf{w}))d\mu(\mathbf{w})} d\mu(\mathbf{w}) = \mathcal{U}_t(S),\end{aligned}$$

where the final equality follows from (3.1). □

We now briefly outline our approach, which follows the lines of [2, 14]. The main technical complication is that sampling efficiently with respect to $\zeta_t$ is not, in general, an easy problem. As a result, we will need some (rather generic) assumption on the investment strategies from which we can sample efficiently.

- *Investment strategies with log-concavity properties.* In section 4.3, we use straightforward manipulations to prove that any investment strategy $S$ which is linear in the vector of parameters $\mathbf{w}$ (such strategies include MA[$k$], SR[$k$], CRP, and CRP-S) has a cumulative return function $\mathcal{R}_t(S(\mathbf{w}))$ that is log-concave. Our efficient sampling techniques are applicable only on such strategies.
- *Approximating $\zeta_t$ by $\bar{\zeta}_t$.* In section 4.2, we show that for strategies whose cumulative return function is log-concave, it is possible to efficiently sample from a distribution $\bar{\zeta}_t$ that is "close" to $\zeta_t$. This "distribution approximation" incurs some small, bounded error (see Lemma 4.2).
- *Approximating the integral for $\bar{\zeta}_t$ via sampling.* With such sampling abilities, it is easy to approximate the average of $S_t(\mathbf{w})$ with respect to $\bar{\zeta}_t$: simply pick $N_t$ (as defined in Lemma 4.3) sample parameter vectors $\mathbf{w}$ with respect to $\bar{\zeta}_t$ and compute their average. The error incurred by this approximation of the average can be bounded in a straightforward manner using Chernoff bounds.
- *Sampling with respect to $\bar{\zeta}_t$.* The critical issue (addressed in section 4.2) is how to pick vectors $\mathbf{w} \in \mathbb{W}$ with respect to $\bar{\zeta}_t$. In order to tackle this problem, we "discretize" it by placing a grid on $\mathbb{W}$, and then we perform a Metropolis random walk. The convergence properties of this random walk are discussed in Theorems 4.12 and 4.13.

In section 4.2, we show that for certain strategies we can efficiently sample from a distribution $\bar{\zeta}_t$ that is "close" to $\zeta_t$; i.e., given $\gamma_t > 0$, we generate samples from $\bar{\zeta}_t$ such that

$$\int_{\mathbb{W}} \left|\zeta_t(\mathbf{w}) - \bar{\zeta}_t(\mathbf{w})\right| d\mu(\mathbf{w}) \le \gamma_t. \tag{4.1}$$

Assume for now that we can sample from $\bar{\zeta}_t$, with $\gamma_t = \frac{\varepsilon^2}{4m(t+1)^4}$, where $\varepsilon$ is the constant appearing in Remark 6. Let $\bar{\mathcal{U}}_t(S) = \int_{\mathbb{W}} S_t(\mathbf{w})d\bar{\zeta}_t(\mathbf{w})$ be the corresponding approximation to $\mathcal{U}(S)$. Lemma 4.2 tells us that we do not lose much by sampling from $\bar{\zeta}_t$.

LEMMA 4.2. *For all $n \geq 0$, (1) $\mathcal{R}_n(\bar{\mathcal{U}}(S)) \geq (1-\varepsilon)\mathcal{R}_n(\mathcal{U}(S))$ and (2) if $\mathcal{U}(S)$ is a universalization of $S$, then $\bar{\mathcal{U}}(S)$ is a universalization of $S$ as well.*

*Proof.* Statement (2) follows directly from (1). To see (1), we need only show that the fraction of wealth that we put into each stock $i$ on day $t$ under $\bar{\mathcal{U}}(S)$ is within a $1-\frac{\varepsilon}{2(t+1)^2}$ factor of the corresponding amount under $\mathcal{U}(S)$; i.e., $\bar{\mathcal{U}}_{ti}(S) \geq (1-\frac{\varepsilon}{2(t+1)^2})\mathcal{U}_{ti}(S)$ for $0 \leq t < n$ and $1 \leq i \leq m$. For $\mathbf{w} \in \mathbb{W}$, let $\gamma_t(\mathbf{w}) = |\bar{\zeta}_t(\mathbf{w}) - \zeta_t(\mathbf{w})|$, so that $\int_{\mathbb{W}} \gamma_t(\mathbf{w})d\mathbf{w} = \gamma_t \leq \frac{\varepsilon^2}{4m(t+1)^4}$. We have

$$\begin{aligned}
\bar{\mathcal{U}}_{ti}(S) &= \int_{\mathbb{W}} S_{ti}(\mathbf{w})\bar{\zeta}_t(\mathbf{w})d\mu(\mathbf{w}) \geq \int_{\mathbb{W}} S_{ti}(\mathbf{w})(\zeta_t(\mathbf{w}) - \gamma_t(\mathbf{w}))d\mu(\mathbf{w}) \\
&= \mathcal{U}_{ti}(S) - \int_{\mathbb{W}} S_{ti}(\mathbf{w})\gamma_t(\mathbf{w})d\mu(\mathbf{w}) \geq \mathcal{U}_{ti}(S) - \gamma_t \qquad (\text{since } S_{ti}(\mathbf{w}) \leq 1) \\
&\geq \left(1 - \frac{\varepsilon}{2(t+1)^2}\right)\mathcal{U}_{ti}(S) \\
&\qquad \left(\text{since } \mathcal{U}_{ti}(S) \geq \min_{\mathbf{w}} S(\mathbf{w}) \geq \frac{\varepsilon}{2m(t+1)^2} \text{ and } \gamma_t \leq \tfrac{\varepsilon^2}{4m(t+1)^4}\right),
\end{aligned}$$

as we wanted to show. ∎

By sampling from $\bar{\zeta}_t$, we use a generalization of the Chernoff bound to get an approximation $\tilde{\mathcal{U}}(S)$ to $\bar{\mathcal{U}}(S)$ such that with high probability $\tilde{\mathcal{U}}_{ti}(S) \geq (1-\frac{\varepsilon}{2(t+1)^2})\bar{\mathcal{U}}_{ti}(S)$ for $0 \leq t < n$ and $1 \leq i \leq m$. Using an argument similar to that in the proof of Lemma 4.2, we see that if $\bar{\mathcal{U}}(S)$ is a universalization of $S$, then such a $\tilde{\mathcal{U}}(S)$ is a universalization of $S$ as well. Choose $\mathbf{w}_1, \ldots, \mathbf{w}_{N_t} \in \mathbb{W}$ at random according to distribution $\bar{\zeta}_t$, and let $\tilde{\mathcal{U}}_{ti}(S) = \frac{1}{N_t}\sum_{i=1}^{N_t} S_{ti}(\mathbf{w}_i)$. Lemma 4.3 discusses the number of samples $N_t$ required to get a sufficiently good approximation to $\bar{\mathcal{U}}_t(S)$.

LEMMA 4.3. *Given $0 < \delta < 1$, use $N_t \geq \frac{8m^2(t+1)^8}{\varepsilon^4}\log\frac{2m(t+1)^2}{\delta}$ samples to compute $\tilde{\mathcal{U}}_t(S)$, where $\varepsilon$ is the constant appearing in Remark 6. With probability $1-\delta$, $\tilde{\mathcal{U}}_{ti}(S) \geq (1-\frac{\varepsilon}{2(t+1)^2})\bar{\mathcal{U}}_{ti}(S)$ for all $1 \leq i \leq m$ and $t \geq 0$.*

*Proof.* Hoeffding [13] proves a general version of the Chernoff bound. For random variables $0 \leq X_i \leq 1$ with $\mathbb{E}(X_i) = \mu$ and $\tilde{X} = \frac{1}{N}\sum_{i=1}^{N} X_i$, the bound states that $\Pr(\tilde{X} \leq (1-\alpha)\mu) \leq e^{-2N\alpha^2\mu^2}$. In our case, we would like $\tilde{\mathcal{U}}_{ti} \geq (1-\frac{\varepsilon}{2(t+1)^2})\bar{\mathcal{U}}_{ti}$. As this must hold for $1 \leq i \leq m$ and $t \geq 0$ with total probability $1-\delta$, we require $\Pr(\tilde{\mathcal{U}}_{ti} \leq (1-\frac{\varepsilon}{2(t+1)^2})\bar{\mathcal{U}}_{ti}) \leq \frac{\delta}{2m(t+1)^2}$ for each $i$ and $t$. From our assumption stated in Remark 6, $\mu = \bar{\mathcal{U}}_{ti} \geq \frac{\varepsilon}{2m(t+1)^2}$, and the desired probability bound is achieved with $N_t \geq \frac{8m^2(t+1)^8}{\varepsilon^4}\log\frac{2m(t+1)^2}{\delta}$ samples. ∎

**4.2. Efficient sampling.** We now discuss how to sample from $\mathbb{W} = \mathcal{W}_k^\ell = \mathcal{W}_k \times \cdots \times \mathcal{W}_k$ according to distribution $\zeta_t(\cdot) \propto \mathcal{R}_t(\cdot) = \mathcal{R}_t(S(\cdot))$. $\mathbb{W}$ is a convex set of diameter $d = \sqrt{2\ell}$. We focus on a discretization of the sampling problem. Choose an orthogonal coordinate system on each $\mathcal{W}_k$, and partition it into hypercubes of side length $\delta_t$, where $\delta_t$ is a constant chosen below. Let $\Omega$ be the set of centers of cubes that intersect $\mathbb{W}$, and choose the partition such that the coordinates of $\mathbf{w} \in \Omega$ are multiples of $\delta_t$. For $\mathbf{w} \in \Omega$, let $C(\mathbf{w})$ be the cube with center $\mathbf{w}$. We show how to choose $\mathbf{w} \in \Omega$ with probability "close to"

$$\pi_t(\mathbf{w}) = \frac{\mathcal{R}_t(\mathbf{w})}{\sum_{\mathbf{w}\in\Omega} \mathcal{R}_t(\mathbf{w})}.$$

In particular, we sample from a distribution $\tilde{\pi}_t$ that satisfies

$$\sum_{\mathbf{w}\in\Omega} |\pi_t(\mathbf{w}) - \tilde{\pi}_t(\mathbf{w})| \leq \gamma_t = \frac{\varepsilon^2}{4m(t+1)^4}. \tag{4.2}$$

Note that this is a discretization of (4.1). We will also have that for each $\mathbf{w} \in \Omega$,

$$\frac{\tilde{\pi}_t(\mathbf{w})}{\pi_t(\mathbf{w})} \leq 2. \tag{4.3}$$

We would like to choose $\delta_t$ sufficiently small that $\mathcal{R}_t$ is "nearly constant" over $C(\mathbf{w})$; i.e., there is a small constant $\nu > 0$ such that

$$(1+\nu)^{-1}\mathcal{R}_t(\mathbf{w}) \leq \mathcal{R}_t(\mathbf{w}') \leq (1+\nu)\mathcal{R}_t(\mathbf{w}) \tag{4.4}$$

for all $\mathbf{w}' \in C(\mathbf{w})$. Such a $\delta_t$ can be chosen for investment strategies $S$ that have bounded derivative, as we see in Lemma 4.4.

LEMMA 4.4. *Suppose that investment strategy $S$ satisfies the condition for universalizability given in Theorem 3.3; i.e., $\left|\frac{\partial S_{ti}(\mathbf{w})}{\partial w_j}\right| \leq c(t+1)$. Given $\nu > 0$, let $\delta_t = \delta_t(\nu) = \frac{\nu}{3c'mt^4k\ell}$, where $c'$ is defined in the proof of Theorem 3.3. For $\mathbf{w}, \mathbf{w}' \in \mathbb{W}$ such that $|w_{ij} - w'_{ij}| \leq \delta_t(\nu)$ for all $1 \leq i \leq \ell$ and $1 \leq j \leq k$, $(1+\nu)^{-1}\mathcal{R}_t(\mathbf{w}) \leq \mathcal{R}_t(\mathbf{w}') \leq (1+\nu)\mathcal{R}_t(\mathbf{w})$.*

*Proof.* Note that $|\mathbf{w} - \mathbf{w}'| \leq \delta_t\sqrt{k\ell}$. Let $\mathbf{w}^*$ be the parameters that maximize the return on the line between $\mathbf{w}$ and $\mathbf{w}'$. By the multivariate mean value theorem and the bound for $|\nabla\mathcal{R}_t|$ given in (3.2),

$$\begin{aligned}
\mathcal{R}_t(\mathbf{w}^*) &= \mathcal{R}_t(\mathbf{w}) + \mathcal{R}_t(\mathbf{w}^*) - \mathcal{R}_t(\mathbf{w}) \\
&\leq \mathcal{R}_t(\mathbf{w}) + |\nabla\mathcal{R}_t(\mathbf{w}_m)| \cdot |\mathbf{w} - \mathbf{w}^*| \quad \text{(for some } \mathbf{w}_m \text{ between } \mathbf{w}^* \text{ and } \mathbf{w}) \\
&\leq \mathcal{R}_t(\mathbf{w}) + c'\mathcal{R}_t(\mathbf{w}_m)mn^4\sqrt{k\ell} \cdot \delta_t\sqrt{k\ell} \leq \mathcal{R}_t(\mathbf{w}) + \mathcal{R}_t(\mathbf{w}^*)\frac{\nu}{3} \\
\Rightarrow \quad \mathcal{R}_t(\mathbf{w}) &\geq \mathcal{R}_t(\mathbf{w}^*)\left(1 - \frac{\nu}{3}\right) \geq \mathcal{R}_t(\mathbf{w}')\left(1 - \frac{\nu}{3}\right)
\end{aligned}$$

so that $\mathcal{R}_t(\mathbf{w}') \leq (1+\nu)\mathcal{R}_t(\mathbf{w})$. By similar reasoning,

$$\begin{aligned}
\mathcal{R}_t(\mathbf{w}') &= \mathcal{R}_t(\mathbf{w}^*) + \mathcal{R}_t(\mathbf{w}') - \mathcal{R}_t(\mathbf{w}^*) \\
&\geq \mathcal{R}_t(\mathbf{w}^*) - |\nabla\mathcal{R}_t(\mathbf{w}_m)| \cdot |\mathbf{w}' - \mathbf{w}^*| \qquad \text{(for some } \mathbf{w}_m \text{ between } \mathbf{w}^* \text{ and } \mathbf{w}') \\
&\geq \mathcal{R}_t(\mathbf{w}^*)\left(1 - \frac{\nu}{3}\right) \geq \mathcal{R}_t(\mathbf{w})\left(1 - \frac{\nu}{3}\right) \geq \mathcal{R}_t(\mathbf{w})(1+\nu)^{-1},
\end{aligned}$$

completing the proof. ∎

We use a Metropolis algorithm [15] to sample from $\tilde{\pi}_t$. We generate a random walk on $\Omega$ according to a Markov chain whose stationary distribution is $\pi_t$. Begin by selecting a point $\mathbf{w}_0 \in \Omega$ according to either $\tilde{\pi}_{t-1}$ or $\tilde{\pi}_{t-2}$;[7] Remark 8 explains how to do this.

[7]Ideally, we would like to begin with a point selected according to $\tilde{\pi}_{t-1}$, but, as discussed in Remark 8, this is not always possible.

REMARK 8. *We can select a point according to $\tilde{\pi}_{t-1}$ by "saving" our samples that were generated at time $t-1$. By Lemma 4.3, we would have generated $N_{t-1} \geq \frac{8m^2t^8}{\varepsilon^4}\log\frac{2mt^2}{\delta}$ samples at time $t-1$, which is not enough to generate the $N_t \geq \frac{8m^2(t+1)^8}{\varepsilon^4}\log\frac{2m(t+1)^2}{\delta}$ samples necessary at time $t$. Instead, we can "save" samples that were generated at times $t-1$ and $t-2$. For sufficiently large $t$, $N_t \leq N_{t-1}+N_{t-2}$ and our initial point $\mathbf{w}_0$ would be picked according to either $\tilde{\pi}_{t-1}$ or $\tilde{\pi}_{t-2}$. As we see in the proof of Lemma 4.10, this distinction is not important.*

If $\mathbf{w}_\tau$ is the position of our random walk at time $\tau \geq 0$, we pick its position at time $\tau+1$ as follows. Note that $\mathbf{w}_\tau$ has $2(k-1)\ell$ neighbors, two along each axis in the Cartesian product of $\ell$ $(k-1)$-dimensional spaces. Let $\mathbf{w}$ be a neighbor of $\mathbf{w}_\tau$, selected uniformly at random. If $\mathbf{w} \in \Omega$, set

$$\mathbf{w}_{\tau+1} = \begin{cases} \mathbf{w} & \text{with probability } p = \min(1, \frac{\mathcal{R}_t(\mathbf{w})}{\mathcal{R}_t(\mathbf{w}_\tau)}), \\ \mathbf{w}_\tau & \text{with probability } 1-p. \end{cases}$$

If $\mathbf{w} \notin \Omega$, let $\mathbf{w}_{\tau+1} = \mathbf{w}_\tau$. It is well known that the stationary distribution of this random walk is $\pi_t$. We must determine how many steps of the walk are necessary before the distribution has gotten sufficiently close to stationary. Let $p_\tau$ be the distribution attained after $\tau$ steps of the random walk. That is, $p_\tau(\mathbf{w})$ is the probability of being at $\mathbf{w}$ after $\tau$ steps.

REMARK 9. *A distinction should be made between $t$ and $\tau$. We use $t$ to refer to the time step in our universalization algorithm. We use $\tau$ to refer to "sub-" time steps used in the Markov chain to sample from $\pi_t$. When $t$ is clear from context, we may drop it from the subscripts in our notation.*

Applegate and Kannan [2] show that if the desired distribution $\pi_t$ is proportional to a log-concave function $F$ (i.e., $\log F$ is concave), then the Markov chain is *rapidly mixing* and reaches its steady state in polynomial time. Frieze and Kannan [9] give an improved upper bound on the mixing time using logarithmic Sobolev inequalities [7].

THEOREM 4.5 (Theorem 1 of [9]). *Assume the diameter $d$ of $\mathbb{W}$ satisfies $d \geq \delta_t\sqrt{k\ell}$ and that the target distribution $\pi$ is proportional to a log-concave function. There is an absolute constant $\kappa > 0$ such that*

$$2\left(\sum_{\mathbf{w}\in\Omega} |\pi(\mathbf{w}) - p_\tau(\mathbf{w})|\right)^2 \leq e^{-\frac{\kappa\tau\delta_t^2}{k\ell d^2}}\log\frac{1}{\pi_*} + \frac{M\pi_e k\ell d^2}{\kappa\delta_t^2}, \tag{4.5}$$

*where $\pi_* = \min_{\mathbf{w}\in\Omega}\pi(\mathbf{w})$, $M = \max_{\mathbf{w}\in\Omega}\frac{p_0(\mathbf{w})}{\pi(\mathbf{w})}\log\frac{p_0(\mathbf{w})}{\pi(\mathbf{w})}$, $p_0(\cdot)$ is the initial distribution on $\Omega$, $\pi_e = \sum_{\mathbf{w}\in\Omega_e}\pi(\mathbf{w})$, and $\Omega_e = \{\mathbf{w} \in \Omega \,|\, \mathrm{Vol}(C(\mathbf{w}) \cap \mathbb{W}) < \mathrm{Vol}(C(\mathbf{w}))\}$. (The "e" in the subscripts of $\pi_e$ and $\Omega_e$ stands for "edge.")*

In the random walk described above, if $\mathbf{w}_\tau$ is on an edge of $\Omega$, so that it has many neighbors outside $\Omega$, the walk may get "stuck" at $\mathbf{w}_\tau$ for a long time, as seen in the "$\pi_e$" term of Theorem 4.5. We must ensure that the random walk has a low probability of reaching such edge points. We do this by applying a "damping function" to $\mathcal{R}_t$, which becomes exponentially small near the edges of $\mathbb{W}$. For $1 \leq i \leq \ell$, $1 \leq j \leq k$, and $\mathbf{w} = (\mathbf{w}_1, \ldots, \mathbf{w}_\ell) = ((w_{11}, \ldots, w_{1k}), \ldots, (w_{\ell 1}, \ldots, w_{\ell k})) \in \mathbb{W}$ let

$$f_{ij}(\mathbf{w}) = e^{\Gamma\min(-\sigma+w_{ij},0)}, \tag{4.6}$$

where $\sigma > 0$ and $\Gamma > 2$ are constants that we choose below, and let

$$F_t(\mathbf{w}) = \mathcal{R}_t(\mathbf{w})\prod_{i=1}^{\ell}\prod_{j=1}^{k} f_{ij}(\mathbf{w}).$$

LEMMA 4.6. *$F_t$ is log-concave if and only if $\mathcal{R}_t$ is log-concave.*[8]

*Proof.* This follows from the fact that log-concave functions are closed under multiplication and the fact that $\log f_{ij}(\mathbf{w}) = \Gamma \min(-\sigma + w_{ij}, 0)$, which is concave. $\square$

Choose $\sigma = \frac{1}{k}\delta_t(\frac{\gamma_t}{2})$, where $\delta_t(\cdot)$ is defined in Lemma 4.4 and $\gamma_t$ is defined in (4.2). Let $\zeta_F \propto F_t$ be the probability measure proportional to $F_t$. We need to show that, for our purposes, sampling from $\zeta_F$ is not much different than sampling from $\zeta_t$. By Lemma 4.2, we can do this by showing that $\int_{\mathbb{W}} |\zeta_t(\mathbf{w}) - \zeta_F(\mathbf{w})| d\mathbf{w} \leq \gamma_t$, which we do in Lemma 4.7.

REMARK 10. *Before continuing, we show how $\mathbb{W}$ can be scaled, which will be useful in future proofs. Take $\mathbf{p} = (\frac{1}{k}, \dots, \frac{1}{k}) \in \mathcal{W}_k$; given $\chi \in (-1, 1)$, let*

$$\mathbf{w}^{(\chi)} = (1 + \chi)(\mathbf{w} - \mathbf{p}) + \mathbf{p},$$

*and let*

$$\mathcal{W}_k^{(\chi)} = \{\mathbf{w}^{(\chi)} \mid \mathbf{w} \in \mathcal{W}_k\}$$

*be a scaled version of $\mathcal{W}_k$ about $\mathbf{p}$, where the scaling factor is $1 + \chi$. To extend this scaling to $\mathbb{W} = \mathcal{W}_k^\ell$, given $\mathbf{w} = (\mathbf{w}_1, \dots, \mathbf{w}_\ell) \in \mathbb{W}$, let $\mathbf{w}^{(\chi)} = (\mathbf{w}_1^{(\chi)}, \dots, \mathbf{w}_\ell^{(\chi)})$ and*

$$\mathbb{W}^{(\chi)} = \{\mathbf{w}^{(\chi)} \mid \mathbf{w} \in \mathbb{W}\}.$$

*A fact we use is that for $1 \leq i \leq \ell$, $1 \leq j \leq k$, and $\mathbf{w} = (\mathbf{w}_1, \dots, \mathbf{w}_\ell) \in \mathbb{W}$,*

$$|w_{ij}^{(\chi)} - w_{ij}| = \left| (1 + \chi)\left(w_{ij} - \frac{1}{k}\right) + \frac{1}{k} - w_{ij} \right| \leq |\chi|.$$

LEMMA 4.7. $\int_{\mathbb{W}} |\zeta_t(\mathbf{w}) - \zeta_F(\mathbf{w})| d\mathbf{w} \leq \gamma_t$.

*Proof.* Let $\mathbb{W}' = \mathbb{W}^{(-k\sigma)}$ be the "scaled-in" version of $\mathbb{W}$, as defined in Remark 10. By Lemma 4.4, since $|w_{ij} - w'_{ij}| \leq k\sigma = \delta_t(\frac{\gamma_t}{2})$ for all $i$ and $j$, $\mathcal{R}_t(\mathbf{w}') \geq \frac{1}{1+\frac{\gamma_t}{2}} \mathcal{R}_t(\mathbf{w})$ and

$$\int_{\mathbb{W}'} \mathcal{R}_t(\mathbf{w}) d\mathbf{w} \geq \frac{1}{1 + \frac{\gamma_t}{2}} \int_{\mathbb{W}} \mathcal{R}_t(\mathbf{w}) d\mathbf{w}. \tag{4.7}$$

Let $\mathbb{W}_{eq} = \{\mathbf{w} \in \mathbb{W} \mid F_t(\mathbf{w}) = \mathcal{R}_t(\mathbf{w})\}$ be the subset of $\mathbb{W}$ where $F_t(\cdot)$ and $\mathcal{R}_t(\cdot)$ are equal; $\mathbb{W}' \subset \mathbb{W}_{eq}$ since, by construction of $\mathbf{w}'$, $w'_{ij} \geq \sigma$ for all $i$ and $j$. Let $\mathbb{W}_+ = \{\mathbf{w} \in \mathbb{W} \mid \zeta_F(\mathbf{w}) \geq \zeta_t(\mathbf{w})\}$ be the subset of $\mathbb{W}$ where $\zeta_F(\cdot)$ is at least $\zeta_t(\cdot)$ and let $\mathbb{W}_- = \mathbb{W} - \mathbb{W}_+$. We bound

$$\int_{\mathbb{W}} |\zeta_F(\mathbf{w}) - \zeta_t(\mathbf{w})| d\mathbf{w} = \int_{\mathbb{W}_+} (\zeta_F(\mathbf{w}) - \zeta_t(\mathbf{w})) d\mathbf{w} + \int_{\mathbb{W}_-} (\zeta_t(\mathbf{w}) - \zeta_F(\mathbf{w})) d\mathbf{w}$$

by bounding $\int_{\mathbb{W}_-} (\zeta_t - \zeta_F)$, which also gives a bound for $\int_{\mathbb{W}_+} (\zeta_F - \zeta_t)$, since

$$\int_{\mathbb{W}_+} (\zeta_F - \zeta_t) = \left(1 - \int_{\mathbb{W}_-} \zeta_F\right) - \left(1 - \int_{\mathbb{W}_-} \zeta_t\right) = \int_{\mathbb{W}_-} (\zeta_t - \zeta_F).$$

[8]We characterize investment strategies for which $\mathcal{R}_t$ is log-concave in Theorem 4.14.

Since $F_t \leq \mathcal{R}_t$, $\int_{\mathbb{W}} F_t \leq \int_{\mathbb{W}} \mathcal{R}_t$ and $\zeta_F(\mathbf{w}) = \frac{F_t(\mathbf{w})}{\int_{\mathbb{W}} F_t} \geq \frac{\mathcal{R}_t(\mathbf{w})}{\int_{\mathbb{W}} \mathcal{R}_t} = \zeta_t(\mathbf{w})$ for $\mathbf{w} \in \mathbb{W}_{eq}$; thus $\mathbb{W}' \subset \mathbb{W}_{eq} \subset \mathbb{W}_+$ and $\mathbb{W}_- \subset \mathbb{W} - \mathbb{W}'$. We have

$$\int_{\mathbb{W}_-} (\zeta_t(\mathbf{w}) - \zeta_F(\mathbf{w}))d\mathbf{w} \leq \int_{\mathbb{W}-\mathbb{W}'} \zeta_t(\mathbf{w})d\mathbf{w} = \frac{\int_{\mathbb{W}-\mathbb{W}'} \mathcal{R}_t(\mathbf{w})d\mathbf{w}}{\int_{\mathbb{W}} \mathcal{R}_t(\mathbf{w})d\mathbf{w}} = 1 - \frac{\int_{\mathbb{W}'} \mathcal{R}_t(\mathbf{w})d\mathbf{w}}{\int_{\mathbb{W}} \mathcal{R}_t(\mathbf{w})d\mathbf{w}}$$
$$\leq 1 - \frac{1}{1+\frac{\gamma_t}{2}} \leq \frac{\gamma_t}{2},$$

where the second-to-last inequality follows from (4.7). This completes the proof. ∎

Henceforth, we are concerned with sampling from $\mathbb{W}$ with probability proportional to $F_t(\cdot)$. We use the Metropolis algorithm described above, replacing $R_t(\cdot)$ with $F_t(\cdot)$; we must refine our grid spacing $\delta_t$ so that (4.4) is satisfied by $F_t$; let $\delta'_t$ be the new grid spacing.

LEMMA 4.8. *Suppose that the conditions of Lemma 4.4 are satisfied. Given $\nu > 0$, let $\delta'_t(\nu) = \delta'_t = \frac{\nu}{3\Gamma c' m t^4 k\ell} = \delta_t(\frac{\nu}{\Gamma})$, where $\Gamma$ appears in (4.6). For $\mathbf{w}, \mathbf{w}' \in \mathbb{W}$ such that $|w_{ij} - w'_{ij}| \leq \delta'_t(\nu)$ for all $1 \leq i \leq \ell$ and $1 \leq j \leq k$, $(1+\nu)^{-1}F_t(\mathbf{w}) \leq F_t(\mathbf{w}') \leq (1+\nu)F_t(\mathbf{w})$.*

*Proof.* By Lemma 4.4, $\mathcal{R}_t(\mathbf{w})$ and $\mathcal{R}_t(\mathbf{w}')$ differ by at most a factor of $1 + \frac{\nu}{\Gamma}$. For each $i$ and $j$, $f_{ij}(\mathbf{w})$ and $f_{ij}(\mathbf{w}')$ differ by at most a factor of $e^{\Gamma\delta'_t(\nu)}$, and hence $\prod_{i=1}^{\ell}\prod_{j=1}^{k} f_{ij}(\mathbf{w})$ and $\prod_{i=1}^{\ell}\prod_{j=1}^{k} f_{ij}(\mathbf{w}')$ differ by at most a factor of $e^{k\ell\Gamma\delta'_t(\nu)} = e^{\frac{\nu}{3c'mt^4}}$. Hence, for $\Gamma \geq 2$ and sufficiently large $t$, $F_t(\mathbf{w})$ and $F_t(\mathbf{w}')$ differ by at most a factor of $1+\nu$. ∎

We are now ready to use Theorem 4.5 to select $\tau$ so that the resulting distribution $p_\tau$ satisfies (4.2) (Theorem 4.12) and (4.3) (Theorem 4.13), with $p_\tau$ in place of $\tilde{\pi}_t$ and $F_t$ in place of $\mathcal{R}_t$. We begin with some preliminary lemmas.

LEMMA 4.9. *There is a constant $\beta > 0$ such that $\log\frac{1}{\pi_*} \leq k\ell\Gamma\sigma + k\ell\log\frac{\beta}{\delta'_t} + t\log\frac{2mt^2}{\varepsilon}$, where $\varepsilon$ is defined in Remark 6.*

*Proof.* Take $\beta$ such that the number of points in $\Omega$ is at most $(\frac{\beta}{\delta'_t})^{(k-1)\cdot\ell}$. For $\mathbf{w}_1, \mathbf{w}_2 \in \Omega$, the ratio of single-day returns on day $t'$ using $\mathbf{w}_1$ and $\mathbf{w}_2$ is

$$\frac{S_{t'}(\mathbf{w}_1)\cdot\mathbf{x}_{t'}}{S_{t'}(\mathbf{w}_2)\cdot\mathbf{x}_{t'}} \geq \frac{\varepsilon}{2m(t'+1)^2},$$

by Remark 6 and Lemma 3.4. The ratio of the cumulative returns up to day $t$ is

$$\frac{\mathcal{R}_t(\mathbf{w}_1)}{\mathcal{R}_t(\mathbf{w}_2)} \geq \Big(\frac{\varepsilon}{2mt^2}\Big)^t,$$

and thus $\frac{\mathcal{R}_t(\mathbf{w})}{\sum_{\mathbf{w}\in\Omega}\mathcal{R}_t(\mathbf{w})} \geq (\frac{\delta'_t}{\beta})^{(k-1)\ell}\big(\frac{\varepsilon}{2mt^2}\big)^t$. Factoring in the maximum dampening effect of the $f_{ij}$, $\pi_* \geq e^{-k\ell\Gamma\sigma}(\frac{\delta'_t}{\beta})^{(k-1)\ell}\big(\frac{\varepsilon}{2mt^2}\big)^t$ and $\log\frac{1}{\pi_*} \leq k\ell\Gamma\sigma + k\ell\log\frac{\beta}{\delta'_t} + t\log\frac{2mt^2}{\varepsilon}$. ∎

LEMMA 4.10. $M \leq 4\big(\frac{2m(t+1)^2}{\varepsilon}\big)^2\log\frac{2m(t+1)^2}{\varepsilon}$.

*Proof.* As stated in Remark 8, the initial distribution is either $p_0 = \tilde{\pi}_{t-1}$ or $\tilde{\pi}_{t-2}$. It turns out that the worst case happens when $p_0 = \tilde{\pi}_{t-2}$. For all $\mathbf{w} \in \Omega$, $\frac{\tilde{\pi}_{t-2}(\mathbf{w})}{\pi_{t-2}(\mathbf{w})} \leq 2$ by (4.3) and the following:

$$\begin{aligned}
\frac{\pi_{t-2}(\mathbf{w})}{\pi_t(\mathbf{w})} &= \frac{F_{t-2}(\mathbf{w})}{\sum_{\mathbf{w}\in\Omega} F_{t-2}(\mathbf{w})} \cdot \frac{\sum_{\mathbf{w}\in\Omega} F_t(\mathbf{w})}{F_t(\mathbf{w})} \\
&\leq \frac{F_{t-2}(\mathbf{w})}{F_t(\mathbf{w})} \cdot \frac{F_t(\mathbf{w}')}{F_{t-2}(\mathbf{w}')} \qquad \left(\text{by Lemma 3.4, where } \mathbf{w}' = \arg\max_{\mathbf{w}\in\Omega} \frac{F_t(\mathbf{w})}{F_{t-2}(\mathbf{w})}\right) \\
&= \frac{\mathcal{R}_{t-2}(\mathbf{w})}{\mathcal{R}_t(\mathbf{w})} \cdot \frac{\mathcal{R}_t(\mathbf{w}')}{\mathcal{R}_{t-2}(\mathbf{w}')} \qquad (\text{since the } \{f_{ij}(\cdot)\}_{i,j} \text{ remain constant with time}) \\
&= \frac{(S_t(\mathbf{w}')\cdot\mathbf{x}_t)(S_{t-1}(\mathbf{w}')\cdot\mathbf{x}_{t-1})}{(S_t(\mathbf{w})\cdot\mathbf{x}_t)(S_{t-1}(\mathbf{w})\cdot\mathbf{x}_{t-1})} \leq \left(\frac{2m(t+1)^2}{\varepsilon}\right)^2,
\end{aligned}$$

where the final inequality follows from the discussion in the proof of Lemma 4.9. This proves the result since $\frac{\tilde{\pi}_{t-2}(\mathbf{w})}{\pi_t(\mathbf{w})} = \frac{\tilde{\pi}_{t-2}(\mathbf{w})}{\pi_{t-2}(\mathbf{w})}\frac{\pi_{t-2}(\mathbf{w})}{\pi_t(\mathbf{w})}$. ∎

LEMMA 4.11. *$\pi_e \leq (1+\nu)^4(1+\frac{\gamma_t}{2})e^{-\Gamma\sigma}$, where $\nu$ appears in the definition of $\delta'_t$ in Lemma 4.8, $\gamma_t$ appears in (4.2), and $\Gamma$ and $\sigma$ appear in (4.6).*

*Proof.* Extend our $\delta'_t$-hypercube partition of $\mathbb{W}$ to the hyperplane containing $\mathbb{W}$, and let $\Psi$ be the set of centers of the hypercubes in this extended partition. For $K \subset \mathbb{R}^{k\ell}$, let $\Psi_K$ be the set of grid points $\mathbf{w} \in \Psi$ such that $C(\mathbf{w}) \cap K \neq \emptyset$, so that $\Omega = \Psi_{\mathbb{W}}$. By Lemma 4.8, for $K \subset \mathbb{W}$,

(4.8)
$$\frac{1}{1+\nu}\sum_{\mathbf{w}\in\Psi_K} F_t(\mathbf{w})\mathrm{Vol}(C(\mathbf{w})\cap K) \leq \int_K F_t(\mathbf{w})d\mathbf{w} \leq (1+\nu)\sum_{\mathbf{w}\in\Psi_K} F_t(\mathbf{w})\mathrm{Vol}(C(\mathbf{w})\cap K).$$

Using the notation of Lemma 4.7, let $\mathbb{W}' = \mathbb{W}^{(-k\sigma)}$ be a "scaled-in" version of $\mathbb{W}$; we showed in Lemma 4.7 that for $\mathbf{w} \in \mathbb{W}'$, $F_t(\mathbf{w}) = \mathcal{R}_t(\mathbf{w})$, and that

$$\int_{\mathbb{W}'} F_t(\mathbf{w})d\mathbf{w} = \int_{\mathbb{W}'} \mathcal{R}_t(\mathbf{w})d\mathbf{w} \geq \frac{1}{1+\frac{\gamma_t}{2}}\int_{\mathbb{W}} \mathcal{R}_t(\mathbf{w})d\mathbf{w}. \tag{4.9}$$

Let $\mathbb{W}'' = \mathbb{W}^{(\delta'_t(\nu))}$ be a "scaled-out" version of $\mathbb{W}$, and extend the domains of $F_t(\cdot)$ and $\mathcal{R}_t(\cdot)$ to $\mathbb{W}''$ by defining $F_t(\mathbf{w}'') = F_t(\bar{\mathbf{w}}'')$ and $\mathcal{R}_t(\mathbf{w}'') = \mathcal{R}_t(\bar{\mathbf{w}}'')$ for $\mathbf{w}'' \in \mathbb{W}'' - \mathbb{W}$, where $\bar{\mathbf{w}}''$ is the point where the line between $\mathbf{w}''$ and $\mathbf{p}^\ell = (\mathbf{p},\dots,\mathbf{p}) \in \mathbb{W}$ intersects the boundary of $\mathbb{W}$. By Lemma 4.8 and the construction of the extension of $\mathcal{R}_t$, $\mathcal{R}_t(\mathbf{w}'') \leq (1+\nu)\mathcal{R}_t(\mathbf{w})$ and

$$\int_{\mathbb{W}''} \mathcal{R}_t(\mathbf{w})d\mathbf{w} \leq (1+\nu)\int_{\mathbb{W}} \mathcal{R}_t(\mathbf{w})d\mathbf{w}. \tag{4.10}$$

By construction of $\mathbb{W}''$, $C(\mathbf{w}) \subset \mathbb{W}''$ for $\mathbf{w} \in \Omega_e$; from the definition of $F_t$ and the choice of $\delta'_t$, $F_t(\mathbf{w}) \leq (1+\nu)e^{-\Gamma\sigma}\mathcal{R}_t(\mathbf{w})$ for $\mathbf{w} \in \Omega_e$. Using these facts,

$$\begin{aligned}
\pi_e = \frac{\sum_{\mathbf{w}\in\Omega_e} F_t(\mathbf{w})}{\sum_{\mathbf{w}\in\Omega} F_t(\mathbf{w})} &\leq \frac{\delta_t^{(k-1)\ell}}{\delta_t^{(k-1)\ell}} \cdot \frac{(1+\nu)e^{-\Gamma\sigma}\sum_{\mathbf{w}\in\Omega_e} \mathcal{R}_t(\mathbf{w})}{\sum_{\mathbf{w}\in\Omega} F_t(\mathbf{w})} \\
&\leq (1+\nu)e^{-\Gamma\sigma}\frac{\sum_{\mathbf{w}\in\Psi_{\mathbb{W}''}} \mathrm{Vol}(C(\mathbf{w})\cap\mathbb{W}'')\mathcal{R}_t(\mathbf{w})}{\sum_{\mathbf{w}\in\Psi_{\mathbb{W}}} \mathrm{Vol}(C(\mathbf{w})\cap\mathbb{W})F_t(\mathbf{w})} \qquad \left(\text{since } \mathrm{Vol}(C(\mathbf{w})) = \delta_t^{(k-1)\ell}\right) \\
&\leq (1+\nu)e^{-\Gamma\sigma}\frac{(1+\nu)\int_{\mathbb{W}''} \mathcal{R}_t(\mathbf{w})d\mathbf{w}}{\frac{1}{(1+\nu)}\int_{\mathbb{W}} F_t(\mathbf{w})d\mathbf{w}} \qquad (\text{by } (4.8)) \\
&\leq (1+\nu)^3 e^{-\Gamma\sigma}\frac{\int_{\mathbb{W}''} \mathcal{R}_t(\mathbf{w})d\mathbf{w}}{\int_{\mathbb{W}'} F_t(\mathbf{w})d\mathbf{w}} \leq (1+\nu)^4\left(1+\frac{\gamma_t}{2}\right)e^{-\Gamma\sigma}
\end{aligned}$$

(by (4.9) and (4.10)). ∎

REMARK 11. *We simplify notation below by using $\mathcal{O}^*(\cdot)$ notation, which ignores logarithmic and constant terms. For our purposes, $f(\cdot) = \mathcal{O}^*(g(\cdot))$ if there exists a constant $C \geq 0$ such that $f(\cdot) = \mathcal{O}(g(\cdot)\log^C(k\ell mt/\varepsilon))$. The values derived above in this notation are $\gamma_t = \mathcal{O}^*(\frac{\varepsilon^2}{mt^4})$, $\delta_t = \mathcal{O}^*(\frac{\nu}{mt^4k\ell})$, $\sigma = \mathcal{O}^*(\frac{\varepsilon^2}{m^2t^8k^2\ell})$, $\delta'_t = \mathcal{O}^*(\frac{\nu}{\Gamma mt^4k\ell})$, $\log\frac{1}{\pi_*} = \mathcal{O}^*(k\ell\Gamma\sigma + t)$, $M = \mathcal{O}^*(\frac{m^2t^4}{\varepsilon^2})$, and $\pi_e = \mathcal{O}^*(e^{-\Gamma\sigma})$.*

THEOREM 4.12. *Letting $\Gamma = \mathcal{O}^*(\frac{1}{\sigma}) = \mathcal{O}^*(\frac{m^2t^8k^2\ell}{\varepsilon^2})$, the random walk reaches a distribution $\tilde{\pi}$ that satisfies (4.2) after $\tau = \mathcal{O}^*(\frac{k^7\ell^6m^6t^{24}}{\kappa\nu^2\varepsilon^4})$ steps.*

*Proof.* We show how to bound the right-hand side of (4.5), where the grid spacing $\delta_t$ has been replaced by $\delta'_t$. The second term, $\frac{M\pi_e k\ell d^2}{\kappa{\delta'_t}^2}$, can be made exponentially small in $\Gamma$ by choosing $\Gamma = \mathcal{O}^*(\frac{1}{\sigma})$. The value of $\tau$ stated in the theorem is large enough to make the first term, $e^{-\frac{\kappa\tau{\delta'_t}^2}{k\ell d^2}}\log\frac{1}{\pi_*}$, exponentially small in $\tau$. □

THEOREM 4.13. *Suppose that the distribution $p_{\tau_0}$ obtained after $\tau_0$ steps satisfies*

$$\sum_{\mathbf{w}\in\Omega} |\pi(\mathbf{w}) - p_{\tau_0}(\mathbf{w})| \leq \gamma_t.$$

*After $\tau'_0 \geq \frac{\tau_0}{\tau_0 - \log\frac{1}{\pi_*} - \log\frac{1}{\gamma_t}}\log\frac{1}{\pi_*} = \mathcal{O}^*(\tau_0(k\ell + t))$ steps, the resulting distribution $p_{\tau'_0}$ satisfies*

$$\max_{\mathbf{w}\in\Omega}\frac{p_{\tau'_0}(\mathbf{w})}{\pi(\mathbf{w})} - 1 \leq 1,$$

*which implies (4.3).*

*Proof.* Let $d(\tau) = \frac{1}{2}\sum_{\mathbf{w}\in\Omega}|\pi(\mathbf{w}) - p_\tau(\mathbf{w})|$ and $\hat{d}(\tau) = \max_{\mathbf{w}\in\Omega}\frac{p_\tau(\mathbf{w})}{\pi(\mathbf{w})} - 1$ so that $d(\tau_0) \leq \frac{1}{2}\gamma_t$. Aldous and Fill [1, (5) and (6)] prove that if $\tau \geq \frac{1}{\lambda}\log\frac{1}{\pi_*}$, then $\hat{d}(\tau) \leq 1$, where $\pi_* = \min_{w\in\Omega}\pi_t(w)$ is as defined in the statement of Theorem 4.5 and $\lambda$ is the second-largest eigenvalue of the steady-state transition matrix $P$ of $\pi_t$.

To prove the bound on $\tau'_0$, we show that $\lambda \geq \frac{\tau_0 - \log\frac{1}{\pi_*} - \log\frac{1}{\gamma_t}}{\tau_0} = 1 - \frac{\log\frac{1}{\pi_*} + \log\frac{1}{\gamma_t}}{\tau_0}$. We do this by appealing to a result from Sinclair [17, Proposition 1(i)], which states that

$$\tau_0 \leq \frac{\log\frac{1}{\pi_*} + \log\frac{1}{\gamma_t}}{1-\lambda}.[9]$$

Solving for $\lambda$ yields the bound for $\tau'_0$. The $\mathcal{O}^*(\cdot)$ bound comes from the fact that $\Gamma\sigma = \mathcal{O}^*(1)$ and that $\log\frac{1}{\gamma_t}$ and $\log\frac{1}{\pi_*}$ are low-order terms relative to the $\tau_0$ obtained in Theorem 4.12. □

**4.3. Application to investment strategies.** The efficient sampling techniques of this section are applicable to investment strategies $S$ whose return functions $\mathcal{R}_n(S(\cdot))$ are log-concave. Theorem 4.14 and Corollary 4.15 characterize such functions.

THEOREM 4.14. *Given investment strategy $S$, suppose that $S$ is linear on $\mathbf{w}$, or, more formally, that for all parameters $w_i$ and $w_j$, $\frac{\partial^2 S}{\partial w_i \partial w_j} = 0$. Then $\mathcal{R}_t(\mathbf{w}) = \mathcal{R}_t(S(\mathbf{w}))$ is log-concave.*

[9]Strictly speaking, this result pertains to $\lambda_{\max}$, the second-largest absolute value of the eigenvalues of $P$, but as Sinclair discusses [17, p. 355], the smallest eigenvalue is unimportant, as $P$ can be modified so that all eigenvalues are positive without affecting mixing times beyond a constant factor.

*Proof.* Let $r_t(\mathbf{w}) = S_t(\mathbf{w}) \cdot \mathbf{x}_t$, so that $\mathcal{R}_n(\mathbf{w}) = \prod_{t=0}^{n-1} r_t(\mathbf{w})$. Since log-concave functions are closed under multiplication, we need only show that $r_t(\mathbf{w})$ is log-concave. The gradient vector of $\log r_t(\mathbf{w})$ has $i$th element $\frac{\partial \log r_t(\mathbf{w})}{\partial w_i} = \frac{1}{r_t(\mathbf{w})} \frac{\partial r_t(\mathbf{w})}{\partial w_i}$, and the matrix of second derivatives has $(i, j)$th element

$$-\frac{1}{r_t(\mathbf{w})^2} \frac{\partial r_t(\mathbf{w})}{\partial w_i} \frac{\partial r_t(\mathbf{w})}{\partial w_j} + \frac{1}{r_t(\mathbf{w})} \frac{\partial^2 r_t(\mathbf{w})}{\partial w_i \partial w_j} = -\frac{1}{r_t(\mathbf{w})^2} \frac{\partial r_t(\mathbf{w})}{\partial w_i} \frac{\partial r_t(\mathbf{w})}{\partial w_j},$$

since $\frac{\partial^2 r_t(\mathbf{w})}{\partial w_i \partial w_j} = \sum_{\iota=1}^{m} \frac{\partial^2 S_{t\iota}(\mathbf{w})}{\partial w_i \partial w_j} \cdot x_{t\iota} = 0$ by assumption. The matrix of second derivatives is negative semidefinite, implying that $\log r_t(\mathbf{w})$ is a concave function. ☐

COROLLARY 4.15. *Universalizations of the following investment strategies can be computed using the sampling techniques of this section:*

1. *the trading strategies MA[k] and SR[k] with long/short allocation functions $g_\ell(x)$ and $h_p(x, y)$, respectively, and*
2. *the portfolio strategies CRP and CRP-S.*

*Proof.* The result follows from a straightforward differentiation of the investment descriptions of these strategies. ☐

**5. Further research.** We have introduced in this paper a general framework for universalizing parameterized investment strategies. It would be interesting to relax the condition of Theorem 3.3 and generalize the theorem. Likewise, it would be interesting to see whether the proof of Theorem 3.3 can be optimized so that existing universal portfolio proofs for CRP [3, 5, 6] are a special case of Theorem 3.3. These proofs not only prove that $\mathcal{L}_n(\mathcal{U}(\text{CRP}))$ converges to $\mathcal{L}_n(\text{CRP}(\mathbf{w}_n^*))$, but also prove a bound on the rate of convergence,

$$\frac{\mathcal{R}_n(\text{CRP}(\mathbf{w}_n^*))}{\mathcal{R}_n(\mathcal{U}(\text{CRP}))} \leq \binom{n+m-1}{m-1} \leq (n+1)^{m-1}.$$

It would also be interesting to study other trading and portfolio strategies that fit into our universalization framework and to see how our universalization algorithms perform in empirical tests.

**Acknowledgments.** We would like to thank two anonymous reviewers for their comments, which significantly improved the presentation of our work.

## REFERENCES

[1] D. Aldous and J. A. Fill, *Advanced $L^2$ techniques for bounding mixing times*, in Reversible Markov Chains and Random Walks on Graphs, unpublished monograph, 1999; available online at stat-www.berkeley.edu/users/aldous/book.html.

[2] D. Applegate and R. Kannan, *Sampling and integration of near log-concave functions*, in Proceedings of the 23rd Annual ACM Symposium on Theory of Computing, New Orleans, LA, 1991, pp. 156–163.

[3] A. Blum and A. Kalai, *Universal portfolios with and without transaction costs*, Machine Learning, 35 (1999), pp. 193–205.

[4] W. Brock, J. Lakonishok, and B. LeBaron, *Simple technical trading rules and the stochastic properties of stock returns*, J. Finance, 47 (1992), pp. 1731–1764.

[5] T. M. Cover, *Universal portfolios*, Math. Finance, 1 (1991), pp. 1–29.

[6] T. M. Cover and E. Ordentlich, *Universal portfolios with side information*, IEEE Trans. Inform. Theory, 42 (1996), pp. 348–363.

[7] P. Diaconis and L. Saloff-Coste, *Logorathmic Sobolev inequalities for finite Markov chains*, Ann. Appl. Probab., 6 (1996), pp. 695–750.

[8] G. B. Folland, *Real Analysis: Modern Techniques and Their Applications*, John Wiley & Sons, New York, 1984.

[9] A. Frieze and R. Kannan, *Log-Sobolev inequalities and sampling from log-concave distributions*, Ann. Appl. Probab., 9 (1999), pp. 14–26.

[10] H. M. Gartley, *Profits in the Stock Market*, Lambert Gann Publishing Company, Pomeroy, WA, 1935.

[11] D. P. Helmbold, R. E. Schapire, Y. Singer, and M. K. Warmuth, *On-line portfolio selection using multiplicative updates*, Math. Finance, 8 (1998), pp. 325–347.

[12] M. Herbster and M. K. Warmuth, *Tracking the best expert*, Machine Learning, 32 (1998), pp. 151–178.

[13] W. Hoeffding, *Probability inequalities for sums of bounded random variables*, J. Amer. Statist. Assoc., 58 (1963), pp. 13–30.

[14] A. Kalai and S. Vempala, *Efficient algorithms for universal portfolios*, J. Mach. Learn. Res., 3 (2002), pp. 423–440.

[15] N. Metropolis, A. W. Rosenbluth, M. N. Rosenbluth, A. H. Teller, and E. Teller, *Equation of state calculation by fast computing machines*, J. Chem. Phys., 21 (1953), pp. 1087–1092.

[16] E. Ordentlich and T. M. Cover, *Online portfolio selection*, in Proceedings of the 9th Annual Conference on Computational Learning Theory, Desanzano sul Garda, Italy, 1996, ACM, New York, pp. 310–313.

[17] A. Sinclair, *Improved bounds for mixing rates of Markov chains and multicommodity flow*, Combin. Probab. Comput., 1 (1992), pp. 351–370.

[18] R. Sullivan, A. Timmermann, and H. White, *Data-snooping, technical trading rules and the bootstrap*, J. Finance, 54 (1999), pp. 1647–1692.

[19] V. Vovk, *Derandomizing stochastic prediction strategies*, Machine Learning, 35 (1999), pp. 247–282.

[20] V. Vovk, *Competitive on-line statistics*, Internat. Statist. Rev., 69 (2001), pp. 213–248.

[21] V. G. Vovk and C. J. H. C. Watkins, *Universal portfolio selection*, in Proceedings of the 11th Conference on Computational Learning Theory, Madison, WI, 1998, pp. 12–23.

[22] R. Wyckoff, *Studies in Tape Reading*, Fraser Publishing Company, Burlington, VT, 1910.

SIAM J. COMPUT. © 2004 Society for Industrial and Applied Mathematics
Vol. 34, No. 1, pp. 23–40

# LABELING SCHEMES FOR FLOW AND CONNECTIVITY[*]

MICHAL KATZ[†], NIR A. KATZ[†], AMOS KORMAN[‡], AND DAVID PELEG[‡]

**Abstract.** This paper studies labeling schemes for flow and connectivity functions. A flow labeling scheme using $O(\log n \cdot \log \hat{\omega} + \log^2 n)$-bit labels is presented for general $n$-vertex graphs with maximum (integral) capacity $\hat{\omega}$. This is shown to be asymptotically optimal. For edge-connectivity, this yields a tight bound of $\Theta(\log^2 n)$ bits. A $k$-vertex connectivity labeling scheme is then given for general $n$-vertex graphs using at most $3\log n$ bits for $k = 2$, $5\log n$ bits for $k = 3$, and $2^k \log n$ bits for $k > 3$. Finally, a lower bound of $\Omega(k \log n)$ is established for $k$-vertex connectivity on $n$-vertex graphs, where $k$ is polylogarithmic in $n$.

**Key words.** labeling schemes, graphs, distributed data structures, flow, vertex-connectivity, edge-connectivity

**AMS subject classifications.** 05C85, 68R10, 05C40, 05C78

**DOI.** 10.1137/S0097539703433912

## 1. Introduction.

### 1.1. Problem and motivation.

Network representations play an extensive role in the areas of distributed computing and communication networks. Their goal is to cheaply store useful information about the network and make it readily and conveniently accessible. This is particularly significant when the network is large and geographically dispersed and information about its structure must be accessed from various local points in it.

The current paper deals with a network representation method based on assigning *informative labels* to the vertices of the network. In most traditional network representations, the names or identifiers given to the vertices contain no useful information, and they serve only as pointers to entries in the data structure, which forms a *global* representation of the network. In contrast, the labeling schemes studied here use more informative and localized labels for the network vertices. The idea is to associate with each vertex a label selected in a such way that will allow us to infer information about any two vertices *directly* from their labels, without using *any* additional information sources. Hence, in essence this method bases the entire representation on the set of labels alone.

Obviously, labels of unrestricted size can be used to encode any desired information including, in particular, the entire graph structure. Our focus is thus on informative labeling schemes using relatively *short* labels (say, of length polylogarithmic in $n$). Labeling schemes of this type were developed in the past for different graph families and for a variety information types, including vertex adjacency [5, 4, 14, 6], distance [17, 11, 10, 8, 13, 9, 1], tree ancestry [3, 12, 2], and various other tree functions, such as center, least common ancestor, separation level, and Steiner weight of a given subset of vertices [18].

[*]Received by the editors April 27, 2003; accepted for publication (in revised form) March 23, 2004; published electronically October 1, 2004. The work of the fourth author was supported in part by grants from the Israel Science Foundation and the Israel Ministry of Science and Art.
http://www.siam.org/journals/sicomp/34-1/43391.html

[†]Department of Mathematics, Bar Ilan University, Ramat Gan, 52900, Israel (nir_michal@hotmail.com).

[‡]Department of Computer Science and Applied Mathematics, The Weizmann Institute of Science, Rehovot, 76100 Israel (amos.korman@weizmann.ac.il, david.peleg@weizmann.ac.il).

The current paper studies informative labeling schemes for flow and connectivity problems. Flow and connectivity information is useful in the decision making process required for various reservation-based routing and connection establishment mechanisms in communication networks, in which it is desirable to have accurate information about the potential capacity of available routes between any two given endpoints. This flow and connectivity information is particularly useful when it represents online the current availability of route capacity in a dynamically changing setting. The methods presented in the current paper are limited to a static graph with fixed topology and edge capacities. Hence, our results constitute only a preliminary step toward handling the full problem in the dynamic setting. Initial studies of the dynamic setting are presented in [16, 15].

**1.2. Labeling schemes.** Let us first formalize the notion of informative labeling schemes. A *vertex-labeling* of the graph $G$ is a function $L$ assigning a label $L(u)$ to each vertex $u$ of $G$. A labeling scheme is composed of two major components. The first is a *marker* algorithm $\mathcal{M}$ which, given a graph $G$, selects a label assignment $L = \mathcal{M}(G)$ for $G$. The second component is a *decoder* algorithm $\mathcal{D}$ which, given a set of labels $\hat{L} = \{L_1, \ldots, L_k\}$, returns a value $\mathcal{D}(\hat{L})$. The time complexity of the decoder is required to be polynomial in its input size.

Let $f$ be a function defined on sets of vertices in a graph. Given a family $\mathcal{G}$ of weighted graphs, an $f$ *labeling scheme* for $\mathcal{G}$ is a marker-decoder pair $\langle \mathcal{M}_f, \mathcal{D}_f \rangle$ with the following property. Consider any graph $G \in \mathcal{G}$, and let $L = \mathcal{M}_f(G)$ be the vertex-labeling assigned by the marker $\mathcal{M}_f$ to $G$. Then for any set of vertices $W = \{v_1, \ldots, v_k\}$ in $G$, the value returned by the decoder $\mathcal{D}_f$ on the set of labels $\hat{L}(W) = \{L(v) \mid v \in W\}$ satisfies $\mathcal{D}_f(\hat{L}(W)) = f(W)$.

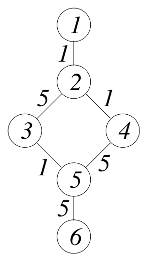
It is important to note that the decoder $\mathcal{D}_f$, responsible for the $f$-computation, is independent of $G$ as well as of the number of vertices in it. Thus $\mathcal{D}_f$ can be viewed as a method for computing $f$-values in a "distributed" fashion, given any set of labels and the knowledge that the graph belongs to some specific family $\mathcal{G}$. In particular, it must be possible to define $\mathcal{D}_f$ as a constant size algorithm. In contrast, the labels contain some information that can be precomputed by considering the whole graph structure.

For a labeling $L$ for the graph $G = \langle V, E \rangle$, let $|L(u)|$ denote the number of bits in the (binary) string $L(u)$. Given a graph $G$ and a marker algorithm $\mathcal{M}$, which assigns the labeling $L$ to $G$, denote $\mathcal{L}_{\mathcal{M}}(G) = \max_{u \in V} |L(u)|$. For a finite graph family $\mathcal{G}$, set $\mathcal{L}_{\mathcal{M}}(\mathcal{G}) = \max\{\mathcal{L}_{\mathcal{M}}(G) \mid G \in \mathcal{G}\}$. Finally, given a function $f$ and a graph family $\mathcal{G}$, let

$$\mathcal{L}(f, \mathcal{G}) = \min\{\mathcal{L}_{\mathcal{M}}(\mathcal{G}) \mid \exists \mathcal{D},\ \langle \mathcal{M}, \mathcal{D} \rangle \text{ is an } f \text{ labeling scheme for } \mathcal{G}\}.$$

**1.3. Flow and connectivity.** In the current paper we focus on flow and connectivity labeling schemes. Let $G = \langle V, E, \omega \rangle$ be a weighted undirected graph where, for every edge $e \in E$, the weight $\omega(e)$ is integral and represents the *capacity* of the edge. For two vertices $u, v \in V$, the *maximum flow* possible between them (in either direction), denoted $\mathtt{flow}(u, v)$, can be defined in this context as follows. Denote by $G'$ the multigraph obtained by replacing each edge $e$ in $G$ with $\omega(e)$ parallel edges of capacity 1. A set of paths $P$ in $G'$ is *edge-disjoint* if each edge $e \in E$ appears in no more than one path $p \in P$. Let $\mathcal{P}_{u,v}$ be the collection of all sets $P$ of edge-disjoint paths in $G'$ between $u$ and $v$. Then $\mathtt{flow}(u, v) = \max_{P \in \mathcal{P}_{u,v}} \{|P|\}$. See Figure 1.

As a special case of the flow function, the *edge-connectivity* $\mathtt{e\text{-}conn}(u, w)$ of two vertices $u$ and $w$ in a graph can be given an alternative definition as the maximum



| | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|
| 1 | $\infty$ | 1 | 1 | 1 | 1 | 1 |
| 2 | | $\infty$ | 6 | 2 | 2 | 2 |
| 3 | | | $\infty$ | 2 | 2 | 2 |
| 4 | | | | $\infty$ | 6 | 5 |
| 5 | | | | | $\infty$ | 5 |
| 6 | | | | | | $\infty$ |

FIG. 1. *A capacitated graph $G$ and the (symmetric) flow between its vertices.*

flow between the two vertices, assuming each edge is assigned one capacity unit.

A set of paths $P$ connecting the vertices $u$ and $w$ in $G$ is *vertex-disjoint* if each vertex, except $u$ and $w$, appears in at most one path $p \in P$. The *vertex-connectivity* $\texttt{v-conn}(u, w)$ of two vertices $u$ and $w$ in an unweighted graph equals the cardinality of the largest set $P$ of vertex-disjoint paths connecting them. By Menger's theorem (cf. [7]), for nonadjacent $u$ and $w$, $\texttt{v-conn}(u, w)$ equals the minimum number of vertices in $G \setminus \{u, w\}$ whose removal from $G$ disconnects $u$ from $w$. (When a vertex is removed, all its incident edges are removed as well.)

**1.4. Our results.** In this paper we present a number of results concerning labeling schemes for maximum flow, edge-connectivity, and vertex-connectivity. In section 2 we present a flow labeling scheme for general graphs, with label size $O(\log n \cdot \log \hat{\omega} + \log^2 n)$ over $n$-vertex graphs with maximum (integral) capacity $\hat{\omega}$. The scheme relies on the fact that the relation "$x$ and $y$ admit a flow of $k$ or more" is an equivalence relation. In section 3 we establish the optimality of our flow labeling scheme by proving a tight lower bound of $\Omega(\log n \cdot \log \hat{\omega} + \log^2 n)$ on the required label size for flow labeling schemes on the class of $n$-vertex trees with maximum capacity $\hat{\omega}$. For edge-connectivity, this yields a tight bound of $\Theta(\log^2 n)$.

In comparison, vertex-connectivity seems to require a more involved labeling scheme whose label size depends on the connectivity parameter $k$. In section 4 we present a $k$-vertex-connectivity labeling scheme for general $n$-vertex graphs. The label sizes we achieve are $\log n$ for $k = 1$, $3 \log n$ for $k = 2$, $5 \log n$ for $k = 3$, and $2^k \log n$ for $k > 3$. In section 5 we present a lower bound of $\Omega(k \log n)$ for the required label size for $k$-vertex connectivity on general $n$-vertex graphs, where $k$ is polylogarithmic in $n$.

## 2. Flow labeling schemes for general graphs.

In this section we consider the family $\mathcal{G}(n, \hat{\omega})$ of undirected capacitated connected $n$-vertex graphs with maximum (integral) capacity $\hat{\omega}$ and present a flow labeling scheme for this family with label size $O(\log n \cdot \log \hat{\omega} + \log^2 n)$. Given a graph $G = \langle V, E, \omega \rangle$ in this family and an integer $1 \le k \le \hat{\omega}$, let us define the following relation:

$$R_k \;=\; \{(x, y) \mid x, y \in V,\;\; \texttt{flow}(x, y) \ge k\}. \tag{2.1}$$

We use the following easy-to-prove fact.

LEMMA 2.1. *The relation $R_k$ is an equivalence relation.*

For every $k \ge 1$, the relation $R_k$ induces a collection of equivalence classes on $V$, $\mathcal{C}_k = \{C_k^1, \ldots, C_k^{m_k}\}$ such that $C_k^i \cap C_k^j = \emptyset$ and $\bigcup_i C_k^i = V$. Note that for $k < k'$, the relation $R_{k'}$ is a *refinement* of $R_k$; namely, for every class $C_{k'}^i$ there is a class $C_k^j$ such that $C_{k'}^i \subseteq C_k^j$.

Given $G$, let us construct a tree $T_G$ corresponding to its equivalence relations. The $k$th level of $T_G$ corresponds to the relation $R_k$, i.e., it has $m_k$ nodes, marked by the classes $C_k^1, \ldots, C_k^{m_k}$. In particular, the root of $T_G$ is marked by the unique equivalence class of $R_1$, which is $V$. The tree is truncated at a node once the equivalence class associated with it is a singleton. For every vertex $v \in G$, denote by $t(v)$ the leaf in $T_G$ associated with the singleton set $\{v\}$. Figure 2 describes the tree $T_G$ corresponding to the flow equivalence classes for the graph $G$ of Figure 1.

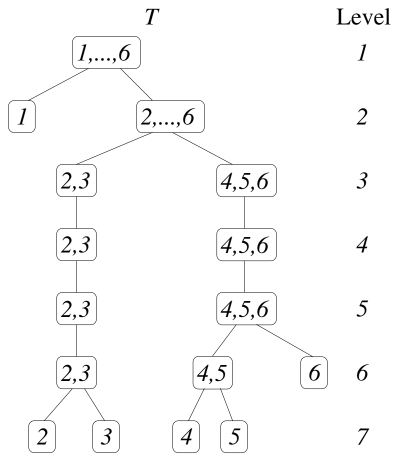


FIG. 2. *The tree $T_G$ corresponding to the graph $G$ of Figure* 1.

For two nodes $x, y$ in a tree $T$ rooted at $r$, define the *separation level* of $x$ and $y$, denoted $\texttt{SepLevel}_T(x, y)$, as the depth of $z = lca(x, y)$, the least common ancestor of $x$ and $y$. In other words, $\texttt{SepLevel}_T(x, y) = dist_T(z, r)$, the distance of $z$ from the root. As an immediate consequence of the construction, we have the following connection.

LEMMA 2.2. *For every two vertices $v, w \in V$,*

$$\texttt{flow}_G(v, w) = \texttt{SepLevel}_T(t(v), t(w)) + 1.$$

It is proven in [18] that for the class $\mathcal{T}(n)$ of $n$-node unweighted trees, there exists a `SepLevel` labeling scheme with $O(\log^2 n)$-bit labels. (This is also shown to be optimal, in the sense that any such scheme must label some node of some $n$-node unweighted tree with an $\Omega(\log^2 n)$-bit label.)

Observe that if the maximum capacity of any edge in the $n$-vertex graph $G$ is $\hat{\omega}$, then the depth of the tree $T_G$ cannot exceed $n\hat{\omega}$ levels; and it may have at most $n$ nodes per level; hence the total number of nodes in $T_G$ is $O(n^2\hat{\omega})$. We immediately have that $\mathcal{L}(\texttt{flow}, \mathcal{G}(n, \hat{\omega})) = O(\log^2(n\hat{\omega}))$.

A more careful design of the tree $T_G$ can improve the bound on the label size. This is achieved by canceling all nodes of degree 2 in the tree $T_G$ and adding appropriate edge weights. Specifically, a subpath $(v_0, v_1, \ldots, v_k)$ in $T_G$ such that $k \geq 2$, $v_0$ and $v_k$ have degree 3 or higher, and $v_1, \ldots, v_{k-1}$ have degree 2 (with $v_1, \ldots, v_k$ all marked by the same set $C$) is compacted into a single edge $(v_0, v_k)$ with weight $k$, eliminating the nodes $v_1, \ldots, v_{k-1}$ and leaving the sets, marking the remaining nodes unchanged. Let $\tilde{T}_G$ denote the resulting compacted tree. Figure 3 describes the tree $\tilde{T}_G$ corresponding to the tree $T_G$ of Figure 2.

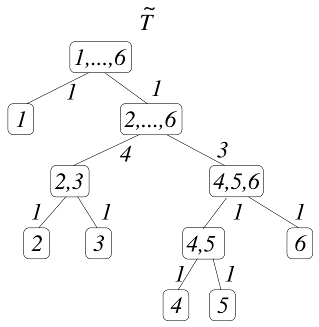


FIG. 3. *The compacted tree $\tilde{T}_G$ corresponding to the tree $T_G$ of Figure* 2.

The notion of separation level can be extended to weighted rooted trees in a natural way by defining $\texttt{SepLevel}_T(x, y)$ as the weighted depth of $z = lca(x, y)$, i.e., its weighted distance from the root. The upper (and lower) bound presented in [18] regarding `SepLevel` labeling schemes for unweighted trees can also be extended in a straightforward manner to weighted trees, yielding `SepLevel` labeling schemes for the class $\mathcal{T}(n, \hat{\omega})$ of weighted $n$-node trees with maximum weight $\hat{\omega}$ using $O(\log n \log \hat{\omega} + \log^2 n)$-bit labels. We give a short overview of this extension.

LEMMA 2.3. $\mathcal{L}(\texttt{SepLevel}, \mathcal{T}(n, \hat{\omega})) \leq \mathcal{L}(distance, \mathcal{T}(n, \hat{\omega})) + \log(n\hat{\omega})$.

*Proof.* Given a distance labeling scheme $\langle \mathcal{M}_{dist}, \mathcal{D}_{dist} \rangle$ for the weighted distance function in $\mathcal{T}(n, \hat{\omega})$, define a `SepLevel` labeling scheme $\langle \mathcal{M}, \mathcal{D} \rangle$ for $\mathcal{T}(n, \hat{\omega})$ as follows. Given a tree $T \in \mathcal{T}(n, \hat{\omega})$, let $L$ be the labeling assigned by $\mathcal{D}_{dist}$ for $T$. The `SepLevel`-marker $\mathcal{M}$ augments each label $L(v)$ into a label $L'(v)$ with an additional $\log(n\hat{\omega})$-bit field containing $v$'s weighted depth, $d(v)$.

For two vertices $x$ and $y$, denote by $d(x, y)$ the weighted distance between $x$ and $y$. Consider two vertices $v, w$ with $z = lca(v, w)$. Let $l_v = d(z, v)$, $l_w = d(z, w)$. Given the labels $L'(v) = \langle L(v), d(v) \rangle$ and $L'(w) = \langle L(w), d(w) \rangle$, the fields $L(v)$ and $L(w)$ allow the `SepLevel`-decoder $\mathcal{D}$ to deduce the weighted distance $d(v, w) = l_v + l_w$, and the two additional fields provide it with $d(v) = l_v + d(z)$ and $d(w) = l_w + d(z)$. Combined, these three equations allow $\mathcal{D}$ to deduce $d(z)$. Thus $\langle \mathcal{M}, \mathcal{D} \rangle$ is a `SepLevel` labeling scheme, and the labels it uses are larger by $\log(n\hat{\omega})$ than those used by $\langle \mathcal{M}_{dist}, \mathcal{D}_{dist} \rangle$. $\square$

Based on the upper bounds of [17, 10] for distance labeling schemes for trees, we get the following.

LEMMA 2.4. $\mathcal{L}(\texttt{SepLevel}, \mathcal{T}(n, \hat{\omega})) = O(\log n \log \hat{\omega} + \log^2 n)$.

It is also easy to verify that for two nodes $x, y$ in $G$, the separation level of the leaves $t(x)$ and $t(y)$ associated with $x$ and $y$ in the tree $\tilde{T}_G$ is still related to the flow between the two vertices, as characterized in Lemma 2.2.

Finally, note that as $\tilde{T}_G$ has exactly $n$ leaves, and every nonleaf node in it has at least two children, the total number of nodes in $\tilde{T}_G$ is $\tilde{n} \leq 2n - 1$. Moreover, the maximum edge weight in $\tilde{T}_G$ is $\tilde{\omega} \leq \hat{\omega} \cdot n$.

Combining the above observations, we have the following.

THEOREM 2.5. $\mathcal{L}(\texttt{flow}, \mathcal{G}(n, \hat{\omega})) = O(\log n \cdot \log \hat{\omega} + \log^2 n)$.

As proved in the next section, this bound is asymptotically optimal.

The above theorem immediately yields the following upper bound for edge-connectivity (which is also shown to be tight in the next section). Let $\mathcal{G}(n)$ denote the class of $n$-vertex unweighted graphs.

COROLLARY 2.6. *$\mathcal{L}(\texttt{e-conn}, \mathcal{G}(n)) = O(\log^2 n)$.*

*Remark.* We note that a similar algorithm applies to any graph function $g$ whose induced relations $R^g_k$, defined as in (2.1), are equivalence relations.

**3. A lower bound for flow labeling schemes on trees.** In this section we establish a lower bound of $\Omega(\log n \cdot \log \hat{\omega} + \log^2 n)$ on the label size for flow on the class $\mathcal{T}(n, \hat{\omega})$ of $n$-vertex trees with maximum edge capacity $\hat{\omega}$ (which is assumed to be integral). The proof idea is based on a modification of the lower bound proof of [10] for distance labeling schemes. Let us first define two more functions on tree vertex pairs, named Max$E$ and Min$E$. For two vertices $u, v$ in a tree $T$, let $Path(u, v)$ denote the unique path from $u$ to $v$ in $T$. Then Max$E(u, v)$ (respectively, Min$E(u, v)$) is the maximum (resp., minimum) weight of an edge on $Path(u, v)$.

Observe that, on a tree, the maximum flow between two vertices $u$ and $v$ equals simply the minimum capacity of an edge on $Path(u, v)$, i.e., $\texttt{flow}(u, v) = \text{Min}E(u, v)$. Hence

$$\mathcal{L}(\texttt{flow}, \mathcal{T}(n, \hat{\omega})) = \mathcal{L}(\text{Min}E, \mathcal{T}(n, \hat{\omega})). \tag{3.1}$$

A Max$E$ labeling scheme $\langle \mathcal{M}, \mathcal{D} \rangle$ can be transformed into a Min$E$ labeling scheme (and vice versa) as follows. Given a weighted tree $T$, let $\hat{\omega}$ denote the maximum weight of an edge in $T$, and let $T'$ be the weighted tree obtained by replacing the weight $\omega(e)$ of every edge $e$ with weight $\omega'(e) = \hat{\omega} - \omega(e)$. The Min$E$ marker $\mathcal{M}'$ will transform $T$ into $T'$ and then apply $\mathcal{M}$. The Min$E$ decoder $\mathcal{D}'$ will invoke $\mathcal{D}$ and then apply the inverse transformation on the resulting weight. As this scheme requires us to encode $\hat{\omega}$ in the labels, we have that

$$\mathcal{L}(\text{Min}E, \mathcal{T}(n, \hat{\omega})) \geq \mathcal{L}(\text{Max}E, \mathcal{T}(n, \hat{\omega})) - \log \hat{\omega}. \tag{3.2}$$

Combining the two relationships (3.1) and (3.2), it follows that, to prove our lower bound on the label sizes required by flow labeling schemes on trees, it suffices to prove it instead for the maximum edge function Max$E$.

We focus on a special subclass of binary weighted trees referred to hereafter as *$(h, \mu)$-trees* for integer $h, \mu \geq 1$. Each tree of this class is a full binary tree with $h$ levels. Number the levels starting from the bottom of the tree, i.e., with the level of the leaves numbered 0. Each edge $e$ is associated with a weight $\omega(e)$ according to its level. The two edges that connect a vertex at level $i + 1$ to its two children at level $i$ are assigned the same weight, taken from the set $Q_i(\mu)$ defined as follows. For $i \geq 0$, let $Z_i(\mu) = i \cdot \mu$ and

$$Q_i(\mu) = \{Z_i(\mu) + j \mid 0 \leq j \leq \mu - 1\}.$$

*Example.* Figure 4 shows a $(3, \mu)$-tree. The weights assigned satisfy $x_{0,i} \in Q_0(\mu)$ for $1 \leq i \leq 4$, $x_{1,i} \in Q_1(\mu)$ for $1 \leq i \leq 2$, and $x_{2,1} \in Q_2(\mu)$.

Note that an $(h, \mu)$-tree $T$ is completely defined by the triple $T = (T_0, T_1, x)$, where $x$ is the weight associated with the two edges of the top level of the tree, and $T_0$ and $T_1$ are the two $(h - 1, \mu)$-trees attached to the endpoints of those two edges. Let $\mathcal{C}(h, \mu)$ be the class of $(h, \mu)$-trees and let $\mathcal{C}(h, \mu, x)$ be the subclass of $\mathcal{C}(h, \mu)$ consisting of $(h, \mu)$-trees with topmost weight $x$. Hence $\mathcal{C}(h, \mu) = \bigcup_{x=0}^{\mu-1} \mathcal{C}(h, \mu, x)$. By the definition of these binary trees we have the following.

OBSERVATION 3.1. *For every two leaves $a, a'$ of a tree $T \in \mathcal{C}(h, \mu, x)$,*

(1) *if $a, a' \in T_i$ (for $i \in \{0, 1\}$), then* $\text{Max}E_T(a, a') = \text{Max}E_{T_i}(a, a')$.

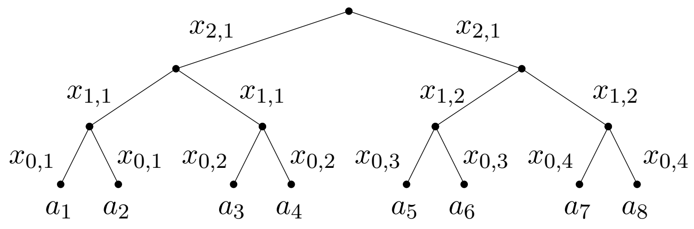


FIG. 4. *A $(3,\mu)$-tree.*

(2) *if $a \in T_0$ and $a' \in T_1$, then* $\mathrm{Max}E_T(a,a') = x$.

This implies the following lemma.

LEMMA 3.2. *Consider two $(h,\mu)$-trees $T = (T_0, T_1, x)$ and $T' = (T'_0, T'_1, x')$. For any leaves $a_0 \in T_0$, $a_1 \in T_1$, $a'_0 \in T'_0$, and $a'_1 \in T'_1$,*

$$\mathrm{Max}E_T(a_0,a_1) \;=\; \mathrm{Max}E_{T'}(a'_0,a'_1) \iff x = x'.$$

We assume that labels are nonnegative integers in $\mathbb{N}$. Define an $(h,\mu)$-*legal* $\mathrm{Max}E$ labeling scheme as a scheme $\langle \mathcal{M}, \mathcal{D}\rangle$ on all binary $(h,\mu)$-trees that correctly provides $\mathrm{Max}E(a_i,a_j)$ between any two leaves $a_i, a_j$. Namely, $\mathcal{D}$ is a decoder function $\mathcal{D} : \mathbb{N}^2 \mapsto \mathbb{N}$, and $\mathcal{M}$ is a marker algorithm assigning a label $L(a,T)$ to each leaf $a$ of any binary $(h,\mu)$-tree $T$, such that for every two leaves $a$ and $a'$ with labels $\lambda = L(a,T)$ and $\lambda' = L(a',T)$, $\mathcal{D}$ computes the maximum weight of an edge between $a$ and $a'$, i.e., $\mathcal{D}(\lambda,\lambda') = \mathrm{Max}E(a,a')$.

For an $(h,\mu)$-*legal* $\mathrm{Max}E$ labeling scheme $\langle \mathcal{M}, \mathcal{D}\rangle$, let $W(\mathcal{M},h,\mu)$ denote the set of all labels assigned by $\mathcal{M}$ to nodes in trees of $\mathcal{C}(h,\mu)$, and let $g(h,\mu)$ denote the minimum cardinality $|W(\mathcal{M},h,\mu)|$ over all flow labeling schemes on $\mathcal{C}(h,\mu)$.

Hereafter, we fix $\langle \hat{\mathcal{M}}, \hat{\mathcal{D}}\rangle$ to be some $\mathrm{Max}E$ labeling scheme attaining $g(h,\mu)$, i.e., such that $|W(\hat{\mathcal{M}},h,\mu)| = g(h,\mu)$.

Let $W(x)$ denote the set of all possible pairs of labels assigned by $\hat{\mathcal{M}}$ to some leaves $a_j \in T_0$ and $a_t \in T_1$, respectively, for some tree $T = (T_0,T_1,x) \in \mathcal{C}(h,\mu,x)$. Let $\mathcal{W} = \bigcup_{x=0}^{\mu-1} W(x)$. As $\mathcal{W} \subseteq W(\hat{\mathcal{M}},h,\mu) \times W(\hat{\mathcal{M}},h,\mu)$, we have the following.

LEMMA 3.3. $|\mathcal{W}| \le g(h,\mu)^2$.

CLAIM 3.4. *For every $0 \le x \ne x' < \mu$, the sets $W(x)$ and $W(x')$ are disjoint.*

*Proof.* Consider two different weights $0 \le x \ne x' < \mu$, and assume by way of contradiction that there exists a pair $(\lambda_1,\lambda_2) \in W(x) \cap W(x')$. Then there exist two $(h-1,\mu)$-trees $T_0, T_1$ such that $T = (T_0,T_1,x)$ uses the label $\lambda_1$ for some leaf $a_{j_1} \in T_0$ and the label $\lambda_2$ for some leaf $a_{j_2} \in T_1$, and there exist two $(h-1,\mu)$-trees $T'_0, T'_1$ such that $T' = (T'_0,T'_1,x')$ uses the label $\lambda_1$ for some leaf $a_{j_3} \in T'_0$ and the label $\lambda_2$ for some leaf $a_{j_4} \in T'_1$. Therefore, by the definition of $\mathcal{D}$,

$$x \;=\; \mathrm{Max}E(a_{j_1},a_{j_2}) \;=\; \mathcal{D}(\lambda_1,\lambda_2) \;=\; \mathrm{Max}E(a_{j_3},a_{j_4}) \;=\; x',$$

implying $x = x'$, a contradiction. □

The following is our main lemma.

LEMMA 3.5. *For every $x \in Q_h(\mu)$, $|W(x)| \ge g(h-1,\mu^2)$.*

*Proof.* In any $(h-1,\mu^2)$-tree, and for every edge that connects a vertex on level $i+1$ to its child on level $0 \le i \le h-1$, a weight $\omega_i \in Q_i(\mu^2)$, $\omega_i = i\cdot\mu^2 + j$, for $0 \le j \le \mu^2 - 1$, can be represented by the pair of weights

$$y_0 \;=\; \omega_i \bmod \mu \;=\; j \bmod \mu \qquad \text{and} \qquad y_1 \;=\; \left\lfloor \frac{\omega_i \bmod \mu^2}{\mu} \right\rfloor \;=\; \left\lfloor \frac{j}{\mu} \right\rfloor,$$

such that $y_0, y_1 \in [0, \mu - 1]$ and $\omega_i = y_0 + \mu y_1 + Z_i(\mu^2)$.

Consequently, one can associate with any $(h-1, \mu^2)$-tree $T'$ a pair of $(h-1, \mu)$-trees $T_0$ and $T_1$ as follows. For any edge $e$ of $T'$ with weight $\omega_i = y_0 + \mu \cdot y_1 + Z_i(\mu^2)$, let the corresponding weight of $e$ in $T_0$ (respectively, $T_1$) be $\omega_{i_0} = Z_i(\mu) + y_0$ (respectively, $\omega_{i_1} = Z_i(\mu) + y_1$). These two trees define also an $(h, \mu)$-tree $T = (T_0, T_1, x)$ in $\mathcal{C}(h, \mu, x)$.

Every leaf $a_j$ of $T'$ is now associated with two homologous leaves of $T$, namely, the leaf $a_j^0 = a_j$ (occurring in the left part of $T$, i.e., $T_0$) and the leaf $a_j^1 = a_{j+2^{h-1}}$ (occurring in $T_1$). For every two leaves $a_j, a_t$ of $T'$ we now have

$$\begin{aligned}\mathrm{Max}E_{T'}(a_j, a_t) &= \mathrm{Max}E_{T_0}(a_j^0, a_t^0) \bmod \mu \\ &\quad + \mu \cdot (\mathrm{Max}E_{T_1}(a_j^1, a_t^1) \bmod \mu) \;+\; \mu^2 \cdot \left\lfloor \frac{\mathrm{Max}E_{T_1}(a_j^1, a_t^1)}{\mu} \right\rfloor \\ &= \mathrm{Max}E_{T}(a_j^0, a_t^0) \bmod \mu \\ &\quad + \mu \cdot (\mathrm{Max}E_{T}(a_j^1, a_t^1) \bmod \mu) \;+\; \mu^2 \cdot \left\lfloor \frac{\mathrm{Max}E_{T}(a_j^1, a_t^1)}{\mu} \right\rfloor .\end{aligned}$$

We use this observation to derive a labeling scheme for all $(h-1, \mu^2)$-trees using at most $|W(x)|$ labels. Given an $(h-1, \mu^2)$-tree $T'$, consider the pair of $(h-1, \mu)$-trees $T_0$, $T_1$ defined above, and use the marker algorithm $\hat{\mathcal{M}}$ to label the tree $T = (T_0, T_1, x)$. Now use the resulting labeling $\hat{L}$ to define a labeling function $L'$ for the nodes of $T'$ as follows. A leaf $a_j \in T'$ receives as its label the pair $\langle L'(a_j, T') = \{\hat{L}(a_j^0, T), \hat{L}(a_j^1, T)\rangle$. Note that this pair belongs to $W(x)$.

The Max$E$ decoder $\mathcal{D}'$ for $(h-1, \mu^2)$-trees, is now obtained by setting

$$\begin{aligned}\mathcal{D}'(L'(a_j, T'), L'(a_t, T')) &= \mathcal{D}'\left(\left\langle \hat{L}(a_j^0, T), \hat{L}(a_j^1, T)\right\rangle, \left\langle \hat{L}(a_t^0, T), \hat{L}(a_t^1, T)\right\rangle\right) \\ &= \hat{\mathcal{D}}(\hat{L}(a_j^0, T), \hat{L}(a_t^0, T)) \bmod \mu \\ &\quad + \mu \cdot \hat{\mathcal{D}}(\hat{L}(a_j^1, T), \hat{L}(a_t^1, T)) \bmod \mu \\ &\quad + \mu^2 \cdot \left\lfloor \frac{\hat{\mathcal{D}}(\hat{L}(a_j^1, T), \hat{L}(a_t^1, T))}{\mu} \right\rfloor .\end{aligned}$$

As $\langle \hat{\mathcal{M}}, \hat{\mathcal{D}} \rangle$ is a Max$E$ labeling scheme for $(h, \mu)$-trees, we have

$$\mathcal{D}(\hat{L}(a_j^0, T), \hat{L}(a_t^0, T)) \;=\; \mathrm{Max}E_T(a_j^0, a_t^0)$$

and

$$\mathcal{D}(\hat{L}(a_j^1, T), \hat{L}(a_t^1, T)) \;=\; \mathrm{Max}E_T(a_j^1, a_t^1);$$

therefore

$$\begin{aligned}\mathcal{D}'(L'(a_j, T'), L'(a_t, T')) &= \mathrm{Max}E_T(a_j^0, a_t^0) \bmod \mu \\ &\quad + \mu \cdot \mathrm{Max}E_T(a_j^1, a_t^1) \bmod \mu \;+\; \mu^2 \cdot \left\lfloor \frac{\mathrm{Max}E_T(a_j^1, a_t^1)}{\mu} \right\rfloor \\ &= \mathrm{Max}E_{T'}(a_j, a_t).\end{aligned}$$

So we have obtained a labeling scheme $\langle \mathcal{M}', \mathcal{D}' \rangle$ labeling any $(h-1, \mu^2)$-tree with labels taken from $W(x)$. It follows that $|W(x)| \geq g(h-1, \mu^2)$. $\square$

Combining Claim 3.4 and Lemmas 3.3 and 3.5, we deduce the following.

Corollary 3.6. $g(h, \mu) \geq \sqrt{\mu} \cdot \sqrt{g(h-1, \mu^2)}$.

Subsequently, we have the following.

Lemma 3.7. $g(h, \mu) \geq \mu^{h/2}$.

This allows us to conclude with the lower bound. Let $\mathcal{BT}(n, \hat{\omega})$ denote the family of $n$-vertex balanced binary trees with height $h = \log(n+1)$ and weights from the range $[0, \hat{\omega}]$, where $\hat{\omega} = h \cdot \mu - 1$.

Theorem 3.8.

$$\mathcal{L}(\text{Max}E, \mathcal{BT}(n, \hat{\omega})) \;\geq\; \frac{1}{2} \cdot \log(n+1) \log(\hat{\omega}+1) - \frac{1}{2} \cdot \log(n+1) \log\log(n+1).$$

*Proof.* By Lemma 3.7, for the class $\mathcal{C}(h, \mu)$ we have $\mathcal{L}(\hat{\omega}, \mathcal{C}(h, \mu)) \geq \frac{h}{2} \cdot \log \mu$. This yields the theorem, as

$$\begin{aligned}\mathcal{L}(\hat{\omega}, \mathcal{BT}(n, \hat{\omega})) \geq \mathcal{L}(\hat{\omega}, \mathcal{C}(h, \mu)) \;&\geq\; \frac{1}{2} h \log \mu \;=\; \frac{1}{2} \log(n+1) \cdot \log\left(\frac{\hat{\omega}+1}{h}\right) \\ &= \frac{1}{2} \cdot \log(n+1) \log(\hat{\omega}+1) - \frac{1}{2} \cdot \log(n+1) \log\log(n+1)\;.\end{aligned}$$

Hence assuming $\hat{\omega} + 1 > \log(n+1)$, there is a lower bound of $\Omega(\log n \log \hat{\omega})$ for the label size of Max$E$ labeling schemes on trees. Finally, by the relationships (3.1) and (3.2) mentioned above between Min$E$ and flow on trees, we get the following.

Corollary 3.9. *For* $\hat{\omega} > \log(n+1) - 1$, $\mathcal{L}(\texttt{flow}, \mathcal{T}(n, \hat{\omega})) = \Omega(\log n \log \hat{\omega})$.

Each tree $T$ of $\mathcal{BT}(n, \hat{\omega})$ can be modified into an unweighted multigraph $G_T$ of $n$ nodes and $O(n\hat{\omega})$ edges by replacing each edge $e$ of weight $\omega(e)$ with $\omega(e)$ parallel edges connecting the same endpoints. This multigraph can in turn be transformed into a simple (unweighted) graph of $O(n\hat{\omega})$ vertices by adding a new vertex $p_{u,v}$ in the middle of every edge $(u, v)$, splitting it into a path of length 2 consisting of the two successive edges $(u, p_{u,v})$ and $(p_{u,v}, v)$. Starting with $\mathcal{BT}(\sqrt{n}, \hat{\omega})$ and looking at the class of unweighted $O(n)$-vertex graphs obtained by setting $\hat{\omega} = \sqrt{n}$, we get the following tight lower bound on the required label size of schemes for flow and edge-connectivity.

Theorem 3.10.

(1) $\mathcal{L}(\texttt{flow}, \mathcal{G}(n)) = \Theta(\log n \log \hat{\omega} + \log^2 n)$.

(2) $\mathcal{L}(\texttt{e-conn}, \mathcal{G}(n)) = \Theta(\log^2 n)$.

## 4. Vertex-connectivity labeling schemes for general graphs.

In this section we turn to $k$-vertex-connectivity and present a labeling scheme for general $n$-vertex graphs. The label sizes we achieve are $\log n$ for $k = 1$, $3 \log n$ for $k = 2$, $5 \log n$ for $k = 3$, and $2^k \log n$ for $k > 3$.

### 4.1. Preliminaries.

We start with some preliminary definitions. In an undirected graph $G$, two vertices are called *$k$-connected* if there exist at least $k$ vertex-disjoint paths between them. A set $S \subseteq V$ *separates* $u$ from $v$ in $G = \langle V, E \rangle$ if $u$ and $v$ are not connected in the vertex-induced subgraph $G \setminus S$.

Theorem 4.1 (Menger (cf. [7])). *In an undirected graph $G$, two nonadjacent vertices $u$ and $v$ are $k$-connected iff no set $S \subset G \setminus \{u, v\}$ of $k-1$ vertices can separate $u$ from $v$ in $G$.*

The *k-connectivity graph* of $G = \langle V, E\rangle$ is $C_k(G) = \langle V, E'\rangle$, where $(u, v) \in E'$ iff $u$ and $v$ are $k$-connected in $G$. A graph $G$ is *closed under k-connectivity* if it has the property that, if $u$ and $v$ are $k$-connected in $G$, then they are neighbors in $G$. Let $\mathcal{C}(k)$ be the family of all graphs $G$ which are closed under $k$-connectivity.

Observation 4.2.

(1) *If $G \in \mathcal{C}(k)$, then each connected component of $G$ belongs to $\mathcal{C}(k)$.*

(2) *If $G = H \cup F$, where $H, F \in \mathcal{C}(k)$ are vertex-disjoint subgraphs of $G$, then $G \in \mathcal{C}(k)$.*

Lemma 4.3. *Let $G' = \langle V, E'\rangle$, where $E' = E \cup \{(u, v)\}$ for some pair of $k$-connected vertices $u$ and $v$. Then $G$ and $G'$ have the same $k$-connectivity graph, i.e., $C_k(G) = C_k(G')$.*

*Proof.* Use induction on $k$. For $k = 1$ the lemma is obvious. Assume the lemma is true for $k - 1$. It suffices to show that if two vertices $w$,$w'$ are not $k$-connected in $G$, then they are not $k$-connected in $G'$. Suppose that $w$,$w'$ are not $k$-connected in $G$. If $w$, $w'$ are neighbors in $G$, then let $G^- = G \setminus \{u, v\}$. In $G^-$, $w$ and $w'$ are not $(k - 1)$-connected and, since $u$ and $v$ are $(k - 1)$ connected in $G^-$, by the induction hypothesis, $w$ and $w'$ are not $(k-1)$-connected in $G' \setminus \{u, v\}$. This implies that they are not $k$-connected in $G'$ as desired. If $w$,$w'$ are not neighbors in $G$, then by Menger's theorem there exists a set of vertices $S = \{x_1, x_2, \ldots, x_{k-1}\}$ that separates $w$ from $w'$ in $G$. We claim that $S$ separates $w$ from $w'$ also in $G'$. The proof breaks down into the following cases.

*Case* 1. One or more of the $x_i$'s is $u$ or $v$. Then $G \setminus S = G' \setminus S$.

*Case* 2. None of the $x_i$'s is $u$ or $v$. If $u$ and $v$ belong to the same connectivity component of $G \setminus S$, then the connectivity components of $G' \setminus S$ will be the same as the connectivity components of $G \setminus S$, implying that $S$ separates $w$ from $w'$ also in $G'$, which is what we wanted to prove. If $u$ and $v$ belong to different connectivity components of $G \setminus S$, then $S$ separates $u$ from $v$ in $G$ or, in other words, $u$ and $v$ are not $k$-connected in $G$, contradicting our assumption. □

Corollary 4.4. *For every graph $G$, if $u$ and $v$ are $k$-connected in $C_k(G)$, then they are neighbors in $C_k(G)$, i.e., $C_k(G) \in \mathcal{C}(k)$.*

*Proof.* Transform a given graph $G$ into $G+ = G \cup C_k(G)$ by adding the edges of $C_k(G)$ to $G$ one by one. By induction on the steps of this process using the previous lemma, we get $C_k(G^+) = C_k(G)$. Therefore if $u$ and $v$ were $k$-connected in $C_k(G)$, then they are $k$-connected in $G^+$ and are therefore neighbors in $C_k(G^+) = C_k(G)$. □

For a connectivity component $C$ of $C_k(G)$, a *leftmost breadth-first search* (*BFS*) tree for $C$, denoted $T(C, k)$, is a *BFS* tree spanning $C$, constructed in the following way. Take a vertex $r$ from $C$ to be the root of $T(C, k)$. Let $level(r) = 1$. Assume we constructed $i$ levels of $T(C, k)$ and haven't used all vertices of $C$. Construct the $(i + 1)$st level of $T(C, k)$ as follows. Repeatedly take a vertex $v$ of level $i$ and connect it to all the vertices adjacent to it in $C_k(G)$ that haven't been included so far in the tree construction. For each such new vertex $w$ let $level(w) = i + 1$, and let $v$ be $w$'s parent in $T(C, k)$.

When the context is clear we use the notation $T$ instead of $T(C, k)$.

For $T = T(C, k)$, we make the following definitions. Let $W_i$ denote the set of vertices of level $i$ in $T$, and let $H_i = H_i(C, k) = \langle W_i, E_i\rangle$ be the subgraph of $C$ induced by $W_i$. For vertices $u$ and $v$, denote $u's$ parent in $T$ by $p(u)$, and let $lca(u, v)$ be the highest level common ancestor of both $u$ and $v$ in $T$. Let $W'_{i+1}$ denote the set of vertices of $W_{i+1}$ that are neighbors of at least $k$ vertices of $W_i$ in $C_k(G)$. Let

$F_i = F_i(C, k)$ be the subgraph of $C$ induced by $W_i \cup W'_{i+1}$.

LEMMA 4.5.

(1) *For* $T = T(C, k)$, $H_i \in \mathcal{C}(k-1)$.

(2) *For* $T = T(C, k)$, $F_i \in \mathcal{C}(k-1)$.

*Proof.* To prove (1), we show that every two vertices $u, v \in W_i$ that are $(k-1)$-connected in $H_i$ are neighbors in $C_k(G)$ and therefore in $H_i$, implying $H_i \in \mathcal{C}(k-1)$. Assume, for contradiction, that $u$ and $v$ are not neighbors in $C_k(G)$. By Corollary 4.4 they are also not $k$-connected in $C_k(G)$; i.e., there exists a set $S = \{x_1, \ldots, x_{k-1}\}$ that separates them in $C_k(G)$. Let $S' = S \cap W_i$. Since $S'$ separates $u$ from $v$ in $H_i$ and since $u$ and $v$ are $(k-1)$-connected in $H_i$, we get that $|S'| = k-1$, and hence $S' = S$, so all the vertices in $S$ must be of level $i$. This implies, however, that $S$ does not separate $u$ from $v$ even in $T$, which is a subgraph of $C_k(G)$, contradicting our assumption.

Turning to (2), let $u$ and $v$ be $(k-1)$-connected in $F_i$. As before, it suffices to show that they are neighbors in $C_k(G)$. Assume for contradiction that $u$ and $v$ are not neighbors in $C_k(G)$. Therefore they are also not $k$-connected in $C_k(G)$; i.e., there exists a set $S = \{x_1, \ldots, x_{k-1}\}$ that separates them in $C_k(G)$. Since $S' = S \cap F_i$ separates $u$ from $v$ in $F_i$ and since $u$ and $v$ are $(k-1)$-connected in $F_i$ we get, as before, that all the vertices of $S$ must belong to $F_i$. We claim that $S$ cannot separate either $u$ or $v$ from the root $r$, and therefore we get that $u$ and $v$ are connected in $C_k(G) \setminus S$, contradicting our assumption. If $u$ (or $v$) is of level $i$, then the claim is clear since $u$ is connected to $r$ in $C_k(G) \setminus S$ via the edges of $T$. If $u$ (or $v$) is of level $i+1$, then $u$ has at least $k$ neighbors in $C_k(G)$ of level $i$. Therefore, $u$ has at least one neighbor $w$ of level $i$ in $C_k(G) \setminus S$. Since all vertices of $S$ are in $F_i$, $w$ is connected to $r$ in $C_k(G) \setminus S$ via the edges of $T$. ☐

**4.2. Overview of the scheme.** We rely on the basic observation that labeling $k$-connectivity for some graph $G$ is equivalent to labeling adjacencies for $C_k(G)$. By Corollary 4.4, $C_k(G) \in \mathcal{C}(k)$. Therefore, instead of presenting a $k$-connectivity labeling scheme for general graphs, we present an adjacency labeling scheme for the graphs of $\mathcal{C}(k)$.

The general idea used for labeling adjacencies for some $G \in \mathcal{C}(k)$, especially for $k > 3$, is to decompose $G$ into at most three "simpler" graphs. One of these graphs is a $k$-orientable graph $K$, and the other two, called $G_{even}$ and $G_{odd}$, belong to $\mathcal{C}(k-1)$. The labeling algorithm for $G \in \mathcal{C}(k)$ recursively labels subgraphs of $G$ that belong to $\mathcal{C}(t)$ for $t < k$. When we are concerned with labeling some $n$-vertex graph $G \in \mathcal{C}(k)$ for $k > 1$, the first step in the labeling is to assign each vertex $u$ in $G$ a distinct identity $id(u)$ from 1 to $n$. This identity will always appear as the last $\log n$ bits of the label $L(G, u)$. Thus, when labeling the subgraphs of $G$ in the recursion we may assume that the *id*'s for the vertices are given.

For graphs $G = \langle V, E\rangle$ and $G_i = \langle V_i, E_i\rangle$, $i > 1$, we say that $G$ can be *decomposed* into the $G_i$'s if $\bigcup_i V_i = V$, $\bigcup_i E_i = E$, and the $E_i$'s are pairwise disjoint.

LEMMA 4.6. *Let* $\mathcal{G}, \mathcal{G}_1$, *and* $\mathcal{G}_2$ *be families of graphs such that each* $G \in \mathcal{G}$ *can be decomposed into* $G_1 \in \mathcal{G}_1$ *and* $G_2 \in \mathcal{G}_2$. *If* $\mathcal{G}_1$ *and* $\mathcal{G}_2$ *have adjacency labeling schemes of sizes* $l_1$ *and* $l_2$, *respectively, then* $\mathcal{G}$ *has adjacency labeling scheme of size* $l_1 + l_2$.

*Proof.* The general idea in the proof is to use concatenation of the labels of the decomposed graphs. Let $\langle \mathcal{M}_i, \mathcal{D}_i\rangle$ be adjacency labeling schemes for $\mathcal{G}_i$ $(i = 1, 2)$. Let us construct an adjacency labeling scheme $\langle \mathcal{M}, \mathcal{D}\rangle$ for $\mathcal{G}$ as follows.

*The marker algorithm* $\mathcal{M}$ *for* $\mathcal{G}$. For a given graph $G \in \mathcal{G}$, decompose $G$ into $G_i \in \mathcal{G}_i$ $(i = 1, 2)$. Let $L_i = \mathcal{M}_i(G_i)$ for $i = 1, 2$. We construct $L = \mathcal{M}(G)$ as follows.

For a vertex $u$ in $G$, let $L(u) = \langle L_1(u), L_2(u)\rangle$, where the first $l_1$ bits of the label $L(u)$ consist of $L_1(u)$ and the next $l_2$ bits give $L_2(u)$. Altogether we use $l_1 + l_2$ bits.

*The decoder algorithm $\mathcal{D}$ for $\mathcal{G}$.* Let $G, G_1$, and $G_2$ be as before. Given the two labels $L(u) = \langle L_1(u), L_2(u)\rangle$ and $L(v) = \langle L_1(v), L_2(v)\rangle$ let $\mathcal{D}(L(u), L(v)) = \mathcal{D}_1(L_1(u), L_1(v)) \vee \mathcal{D}_2(L_2(u), L_2(v))$.

Since $G$ was decomposed into $G_1, G_2$, the vertices $u$ and $v$ are neighbors in $G$ iff they are neighbors in either $G_1$ or $G_2$; hence the decoding algorithm is correct. □

COROLLARY 4.7. *Let $\mathcal{G}, \mathcal{G}_1, \ldots, \mathcal{G}_m$ be families of graph such that each $G \in \mathcal{G}$ can be decomposed into $G_1, \ldots, G_m$, where $G_i \in \mathcal{G}_i$ for $i = 1$ to $m$. If the $\mathcal{G}_i$'s have adjacency labeling schemes of size $l_i$, then $\mathcal{G}$ has an adjacency labeling scheme of size $\sum l_i$.*

A graph $G$ is called *$k$-orientable* if there exists an orientation of the edges such that the out-degree of each vertex is bounded above by $k$. The class of $k$-orientable graphs is denoted $\mathcal{J}_{or}(k)$.

OBSERVATION 4.8. *If $G = H \cup F$, where $H, F \in \mathcal{J}_{or}(k)$ are vertex-disjoint subgraphs of $G$, then $G \in \mathcal{J}_{or}(k)$.*

LEMMA 4.9. *Let $\mathcal{J}_n(k)$ be the family of $n$-vertex graphs in $\mathcal{J}_{or}(k)$. Assuming id's are given, $\mathcal{L}(\texttt{adjacency}, \mathcal{J}_n(k)) \leq k \log n$.*

*Proof.* Suppose $G \in \mathcal{J}_n(k)$. Then $G$ is a $k$-orientable graph with $n$ vertices. Hence there exists an orientation to the edges of $G$ such that the out-degree of each vertex is bounded above by $k$. In this orientation, for each $u$ there exist at most $k$ outgoing edges, say $(u, v_1)$, $(u, v_2)$,..., $(u, v_t)$, for $t \leq k$.

*The marker algorithm $\mathcal{M}$ for $\mathcal{J}_n(k)$.* Label $u$ by $L(u) = \langle id(v_1), id(v_2), \ldots, id(v_t)\rangle$, i.e., use the first $\log n$ bits to write $id(v_1)$, the second $\log n$ bits to write $id(v_2)$, etc. Hence, for every $u$'s, the size of $L(u)$ is at most $k \log n$ bits.

*The decoder algorithm $\mathcal{D}$ for $\mathcal{J}_n(k)$.* Given $L(u)$ and $L(v)$ check whether $u$'s id appears in $L(v)$, by inspecting each block of $\log n$ bits in $L(v)$ separately. Analogously, check if $v$'s id appears in $L(u)$.

As $u$ and $v$ are neighbors in $G$ iff one of the two cases applies, the decoding algorithm is correct. □

To illustrate the approach, we preface the treatment of the general case with a discussion of the cases $k = 1, 2, 3$, for which slightly better schemes are available. The simple case of $k = 1$ is handled in section 4.2.1. For $k = 2$ we show in section 4.2.2 that a connected graph $G \in \mathcal{C}(2)$ can be decomposed into a tree and disjoint graphs in $\mathcal{C}(1)$. Graphs in $\mathcal{C}(1)$ are collections of cliques. It follows that each $G \in \mathcal{C}(2)$ can be decomposed into a forest (which is a 1-orientable graph) and a graph made of disjoint cliques. For $k = 3$ we show in section 4.2.3 that a connected graph $G \in \mathcal{C}(3)$ can be decomposed into a graph in $\mathcal{C}(2)$ and a 2-orientable graph.

#### 4.2.1. A 1-connectivity labeling scheme.

Let us give a labeling scheme for 1-connectivity for $\mathcal{G}_n$, the family of all $n$-vertex graphs.

*The marker algorithm $\mathcal{M}$ for $\mathcal{G}_n$.* Fix $G = \langle V, E\rangle \in \mathcal{G}_n$. To each connected component $C$ of $G$ assign a distinct identity $id(C)$ from the range $\{1, \ldots, n\}$. For a vertex $u \in V$, let $C_u$ be the connected component of $G$ to which $u$ belongs. The marker algorithm sets $L(u) = id(C_u)$.

*The decoder $\mathcal{D}$ for $\mathcal{G}_n$.* Let $D(L(u), L(v)) = 1$ iff $L(u) = L(v)$.

Clearly $u$ and $v$ are 1-connected in $G$ iff they are in the same connected component; hence the decoder's response is correct. The size of the label is bounded above by $\log n$.

THEOREM 4.10. $\mathcal{L}(1-\texttt{v-conn}, \mathcal{G}_n) \leq \log n$.

**4.2.2. A 2-connectivity labeling scheme.** As explained earlier, labeling 2-connectivity for a family of graphs $\mathcal{G}$ is equivalent to labeling adjacencies for the family $\{C_2(G) \mid G \in \mathcal{G}\} \subseteq \mathcal{C}(2)$. In this section we present an efficient adjacency labeling scheme for $\mathcal{C}(2)$.

Consider a graph $G \in \mathcal{C}(2)$ and let $C_1, \ldots, C_m$ be its connected components. By Observation 4.2(1), $C_i \in \mathcal{C}(2)$ for every $i$. Fix $i$ and let $T = T(C_i, 2)$.

CLAIM 4.11. *The only neighbor of $u$ in $G$ which has a strictly lower level than $u$ in $T$ is $p(u)$.*

*Proof.* Suppose, for contradiction, that there exist neighbors $v$ and $w$ such that $level(w) > level(v)$ but $v$ is not $p(w)$ in $T$. In this case, $w$ and $z = lca(v, w)$ are 2-connected in $T \cup \{(v, w)\}$, which is a subgraph of $G$. Since $G \in \mathcal{C}(2)$, $v$ must be a neighbor of $w$. Since $level(w) < level(u) - 1$ we get a contradiction to the way $T$ was constructed. □

CLAIM 4.12. *$G \in \mathcal{C}(2)$ can be decomposed into a forest $F$ and a graph $H$ of disjoint cliques.*

*Proof.* Fix a connected component $C_i$ of $G$ and let $T = T(C_i, 2)$. Since, by Lemma 4.5(1), each $H^i_j$ subgraph of $C_i$ induced by level $j$ of $T$ is in $\mathcal{C}(1)$, it follows that $H^i_j$ is a collection of disjoint cliques. Hence $G$ can be decomposed into a forest $F$ and a graph $H$ of disjoint cliques, composed of the collection of all the $H^i_j$ from all $i$'s and $j$'s. □

Let $\mathcal{C}_n(2)$ be the family of $n$-vertex graphs in $\mathcal{C}(2)$. Let us now give an adjacency labeling scheme for the graphs of $\mathcal{C}_n(2)$.

*The marker algorithm $\mathcal{M}$ for $\mathcal{C}_n(2)$.* Decompose $G$ into $F$ and $H$ as in Claim 4.12. Fix a vertex $u$ of $G$. Let $p(u)$ be $u$'s parent in $F$. To each clique $C$ in $H$ give a distinct identity from the range $\{1, \ldots, n\}$, $id(C)$. Let $C(u)$ be the clique in $H$ that contains $u$.

The marker algorithm for $G$ assigns $L(u) = \langle id(c(u)), id(p(u)), id(u)\rangle$. As before, we use the first $\log n$ bits for $id(c(u))$, the second $\log n$ bits for $id(p(u))$, etc. The label size is bounded above by $3 \log n$.

*The decoder $\mathcal{D}$ for $\mathcal{C}_n(2)$.* Given $L(u)$ and $L(v)$ we compare $id(p(u))$ with $id(v)$ and $id(p(v))$ with $id(u)$ to check whether one is the parent of the other in the forest $F$. We also check if $id(C(u)) = id(C(v))$ to see whether $u$ and $v$ are neighbors in $H$. We do this by looking at the corresponding bits in the label; for example, $id(p(u))$ is written in the second block of $\log n$ bits of $L(u)$. Let $\mathcal{D}(L(u), L(v)) = 1$ iff either $id(C(u)) = id(C(v))$, $id(p(u)) = id(v)$, or $id(p(v)) = id(u)$.

Clearly, $u$ and $v$ are neighbors in $G$ iff they are neighbors in $F$ or in $H$; hence the decoder's response is correct. We get the following.

THEOREM 4.13. *Let $\mathcal{G}_n$ be the family of $n$-vertex graphs. Then $\mathcal{L}(2\text{−}\texttt{v-conn}, \mathcal{G}_n) \leq 3 \log n$ bits.*

**4.2.3. A 3-connectivity labeling scheme.** Again, labeling 3-connectivity for a family $\mathcal{G}$ is equivalent to labeling adjacencies for the family $\{C_3(G) \mid G \in \mathcal{G}\} \subseteq \mathcal{C}(3)$. In this section we show how to label adjacencies for $\mathcal{C}(3)$.

Consider a graph $G \in \mathcal{C}(3)$, and let $C_1, \ldots, C_m$ be its connected components. By observation 4.2(1), $C_i \in \mathcal{C}(3)$ for all $i$. Fix $i$ and let $T = T(C_i, 3)$.

LEMMA 4.14. *Each vertex $u$ has at most one neighbor of $G$ which has a strictly lower level than $u$ in $T$ apart from $p(u)$.*

*Proof.* Assume, for contradiction, that there exists a vertex $u$ with two neighbors in $G$, $v$ and $w$, both with a strictly lower level than $u$ and both different from $p(u)$. In this case, $u$ must be 3-connected in $G$ to either $lca(u, v)$, $lca(u, w)$, or $lca(v, w)$, whichever has the highest level number. However, the levels of $lca(u,v)$, $lca(u,w)$, $lca(v,w)$ are all smaller than $level(u)-1$, and since $G \in \mathcal{C}(3)$, $u$ is adjacent to one of them, contradicting the way $T$ was constructed. (See Figure 5.) ∎

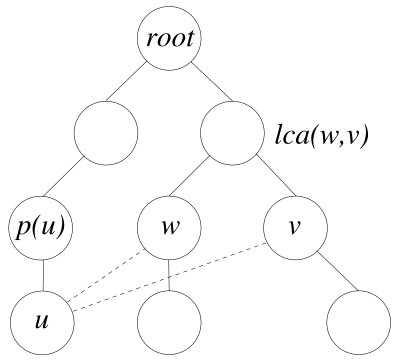


FIG. 5. *An illustration of the contradiction in the proof of Lemma* 4.14.

LEMMA 4.15. *Each $G \in \mathcal{C}(3)$ can be decomposed into a graph $H \in \mathcal{C}(2)$ and a* 2*-orientable graph.*

*Proof.* First, it suffices to show the lemma for connected graphs $C \in \mathcal{C}(3)$, since by Observations 4.2(2) and 4.8, $\mathcal{C}(2)$ and $\mathcal{J}_{or}(2)$ are closed under vertex-disjoint unions. Consider a connected graph $C \in \mathcal{C}(3)$ and let $T = T(C,3)$. By Lemma 4.5(1), each subgraph $H_j$ of $C$ induced by the vertices of level $j$ in $T$ is in $\mathcal{C}(2)$. All the subgraphs $H_j$ are vertex-disjoint; hence by letting $H$ be the union of all the $H_j$, we get $H \in \mathcal{C}(2)$. Let $U$ be the graph $C$ after deleting the edges of $H$. By Lemma 4.14, each vertex $u$ of $U$ has at most two neighbors of a strictly lower level (one of which is $u$'s parent in $T$). Hence directing the edges of $U$ from higher level vertices to lower level vertices, each $u$ has out-degree at most 2; i.e., $U$ is 2-orientable. ∎

By Lemmas 4.6 and 4.9 and from Theorem 4.13 we get the following theorem.

THEOREM 4.16. *Let $\mathcal{G}_n$ be the family of $n$-vertex graphs. Then $\mathcal{L}(3-\texttt{v-conn}, \mathcal{G}_n) \leq 5\log n$ bits.*

## 4.3. A $k$-connectivity labeling scheme.

Finally, labeling $k$-connectivity for a family $\mathcal{G}$ is equivalent to labeling adjacencies for the family $\{C_k(G) \mid G \in \mathcal{G}\} \subseteq \mathcal{C}(k)$. In this section we show how to label adjacencies for $\mathcal{C}(k)$.

Consider a graph $G \in \mathcal{C}(k)$, and let $C_1, \ldots, C_m$ be its connected components. By observation 4.2(1), $C_i \in \mathcal{C}(k)$ for all $i$. Fix $i$ and let $T = T(C_i, k)$.

LEMMA 4.17. *Each $G \in \mathcal{C}(k)$ can be decomposed into two graphs in $\mathcal{C}(k-1)$ and a $(k-1)$-orientable graph.*

*Proof.* Again, it suffices to prove the lemma for connected graphs $C \in \mathcal{C}(k)$ since, by Observations 4.2(2) and 4.8, both $\mathcal{C}(k-1)$ and $\mathcal{J}_{or}(k)$ are closed under vertex-disjoint unions. Consider a connected graph $C \in \mathcal{C}(k)$ and let $T = T(C,k)$.

All the $F_i$'s for odd $i$'s are vertex-disjoint, and $F_i \in \mathcal{C}(k-1)$ for all $i$'s by Lemma 4.5(2). Therefore, by letting $G_{odd}$ be the union of all the $F_i$'s for odd $i$'s, we get $G_{odd} \in \mathcal{C}(k-1)$. By the same reasoning, by letting $G_{even}$ be the union of all the $F_i$'s for even $i$'s, we get $G_{even} \in \mathcal{C}(k-1)$.

Let $K$ be the graph $C$ after omitting the edges of $G_{odd}$ and $G_{even}$ (or equivalently, omitting all edges of all the $F_i$'s). The proof is completed once we show that $K$ is $(k-1)$-orientable. Since all edges $(u,v)$ of $C$ such that $level(u) = level(v) = i$ for some

$i$ are in $F_i$ for the appropriate $i$, if $(u,v)$ is an edge of $K$, then $level(u) \neq level(v)$. By the way $T$ was constructed, we can see that the difference between the levels is 1.

Let us direct the edges of $K$ from higher level vertices to lower level vertices. Assume, for contradiction, that for some $u$ and some $i$, $level(u) = i+1$ and the out-degree of $u$ in $K$ is at least $k$. Then $u$ must have at least $k$ neighbors of level $i$ in $C$, in which case all edges $(u,v)$ for $v$ such that $level(v) = i$ appear in $F_i$ and therefore not in $K$. Thus, the out-degree of $u$ in $K$ is 0, contradicting our assumption. ∎

Before stating and proving the next theorem, let us remark that we can get a weaker upper bound of $3^k \log n$ label size for $\mathcal{L}(\mathtt{adjacency}, \mathcal{C}_n(k))$ in the following way. Use induction on $k$. For $k = 1,2,3$ our remark holds. For $k > 3$ fix $k$ and assume that the remark holds for $k-1$. The remark for $k$ follows from Lemmas 4.9 and 4.17 and Corollary 4.7.

To prove the next theorem, we show that instead of concatenating $u$'s labels in the three decomposed graphs $(G_{odd}, G_{even}, K)$, it suffices to give $u$ its label in only two of the three decomposed graphs. This yields the desired $2^k \log n$ bits bound on $\mathcal{L}(\mathtt{adjacency}, \mathcal{C}_n(k))$.

THEOREM 4.18. *Let $\mathcal{C}_n(k)$ be the family of $n$-vertex graphs in $\mathcal{C}(k)$. Then*

$$\mathcal{L}(\mathtt{adjacency}, \mathcal{C}_n(k)) \;\leq\; 2^k \log n.$$

*Proof.* Use induction on $k$. For $k = 1$, 2, or 3 the theorem holds as seen in Theorems 4.10, 4.13, and 4.16. For $k > 3$, fix $k$ and assume that the theorem holds for $k-1$. Consider a graph $G \in \mathcal{C}_n(k)$. For a vertex $u$ in $G$, let $C$ be its connected component in $G$, let $T = T(C,k)$, and let $i = level(u)$. Let us now give a labeling scheme for adjacency on $G \in \mathcal{C}(k)$.

*The marker algorithm $\mathcal{M}_k$ for $\mathcal{C}(k)$.* For $t \leq k$, $G \in \mathcal{C}_n(t)$, and $u$, a vertex of $G$, denote the adjacency labeling on $G$ by $L_t(G)$ and $u$'s label by $L_t(G,u)$. Let $G \in \mathcal{C}_n(k)$ and let $u$ be a vertex in $G$. We define $\mathsf{State}(u)$ according to the following three cases.

*Case* 1. $u$ participates in both $G_{odd}$ and $G_{even}$. Let $\mathsf{State}(u) = \mathsf{Dual}$. Note that in this case the out-degree of $u$ in $K$ is 0. The marker algorithm assigns to $u$ the label $L_k(G,u) = \langle L_{k-1}(G_{odd},u), L_{k-1}(G_{even},u)\rangle$, where the first $2^{k-1}\log n$ bits are reserved for $L_{k-1}(G_{odd},u)$ and the last $2^{k-1}\log n$ bits are reserved for $L_{k-1}(G_{even},u)$.

*Case* 2. $u$ doesn't participate in $G_{odd}$; i.e., $u$ participates only in $G_{even}$ and in $K$. Let $\mathsf{State}(u) = \mathsf{Even}$. Let $L_k(G,u) = \langle 0^{k\log n}, 10, L(u,K), 00\ldots000, L_{k-1}(G_{even},u)\rangle$, where the two bits in the second field, 10, indicate that $\mathsf{State}(u) = \mathsf{Even}$. The next $k \log n$ bits are reserved for $L(u,K)$ and the last $2^{k-1}\log n$ bits are reserved for $L_{k-1}(G_{even},u)$.

*Case* 3. $u$ doesn't participate in $G_{even}$; i.e., $u$ participates only in $G_{odd}$ and in $K$. Let $\mathsf{State}(u) = \mathsf{Odd}$. Let $L_k(G,u) = \langle 0^{k\log n}, 11, L(u,K), 00\ldots00, L_{K-1}(G_{odd},u)\rangle$, where the two bits in the second field, 11, indicate that $\mathsf{State}(u) = \mathsf{Odd}$, the next $k\log n$ bits are reserved for $L(u,K)$, and the last $2^{k-1}\log n$ bits are reserved for $L_{K-1}(G_{odd},u)$.

The size of the label $L_k(G,u)$ is, by induction, at most $2^k \log n$ since, by Lemma 4.9, the size of $L(v,K)$ is at most $k\log n$ and both sizes of $L_{k-1}(G_{odd},u)$ and $L_{k-1}(G_{even},u)$ are at most $2^{k-1}\log n$.

By the definition of $K$, it is clear that the out-degree of some $u$ in $K$ is higher than 0 iff $\mathsf{State}(u) = \mathsf{Even}$ or $\mathsf{Odd}$.

*The decoder $\mathcal{D}_k$ for $\mathcal{C}(k)$.* For $t \leq k$ denote the decoder for $\mathcal{C}(t)$ by $\mathcal{D}_t$. Denote the decoder for $\mathcal{J}_{or}(k)$ (from Lemma 4.9) by $\mathcal{D}_{or}$. Given $L_k(G,u)$ and $L_k(G,v)$ we will first want to know the states of $u$ and $v$. Take for example $L_k(G,u)$. For $k > 3$, the first $k \log n$ bits are 0 iff $\mathsf{State}(u) \neq \mathsf{Dual}$. So by looking at the first $k \log n + 2$ bits of $L_k(G,u)$ and $L_k(G,v)$ we know the states of $u$ and $v$. Consider the following cases.

*Case* a. $\mathsf{State}(u) = \mathsf{State}(v) = \mathsf{Dual}$: Then $\mathcal{D}_k$ for $G$ uses $\mathcal{D}_{k-1}$ on $G_{even}$ and $G_{odd}$ as follows:

$$\begin{aligned}&\mathcal{D}_k(L_k(G,u), L_k(G,v))\\&= (\mathcal{D}_{k-1}(L_{k-1}(G_{odd},u), L_{k-1}(G_{odd},v)))\\&\vee(\mathcal{D}_{k-1}(L_{k-1}(G_{even},u), L_{k-1}(G_{even},v))).\end{aligned}$$

*Case* b. $\mathsf{State}(u) = \mathsf{State}(v) = \mathsf{Even}$. Then $\mathcal{D}_k$ for $G$ uses $\mathcal{D}_{k-1}$ on $G_{even}$ and $\mathcal{D}_{or}$ for $K$ as follows:

$$\begin{aligned}&\mathcal{D}_k(L_k(G,u), L_k(G,v))\\&= \mathcal{D}_{or}(L(u,K), L(v,K)) \vee \mathcal{D}_{k-1}(L_{k-1}(G_{even},u), L_{k-1}(G_{even},v)).\end{aligned}$$

*Case* c. $\mathsf{State}(u) = \mathsf{State}(v) = \mathsf{Odd}$. Then $\mathcal{D}_k$ for $G$ uses $\mathcal{D}_{k-1}$ on $G_{odd}$ and $\mathcal{D}_{or}$ for $K$ as follows:

$$\begin{aligned}&\mathcal{D}_k(L_k(G,u), L_k(G,v))\\&= \mathcal{D}_{or}(L(u,K), L(v,K)) \vee \mathcal{D}_{k-1}(L_{k-1}(G_{odd},u), L_{k-1}(G_{odd},v)).\end{aligned}$$

*Case* d. $\mathsf{State}(u) = \mathsf{Dual}$, $\mathsf{State}(v) = \mathsf{Even}$. Then let $\mathcal{D}_k(L_k(G,u), L_k(G,v)) = 1$ iff $\mathcal{D}_{k-1}(L_{k-1}(G_{even},u), L_{k-1}(G_{even},v)) = 1$ or $id(u)$ appears in $L(v,K)$.

*Case* e. $\mathsf{State}(u) = \mathsf{Dual}$, $\mathsf{State}(v) = \mathsf{Odd}$. Then let $\mathcal{D}_k(L_k(G,u), L_k(G,v)) = 1$ iff $\mathcal{D}_{k-1}(L_{k-1}(G_{odd},u), L_{k-1}(G_{odd},v)) = 1$ or $id(u)$ appears in $L(v,K)$.

*Case* f. $\mathsf{State}(u) = \mathsf{Even}$, $\mathsf{State}(v) = \mathsf{Odd}$. Then

$$\mathcal{D}_k(L_k(G,u), L_k(G,v)) = \mathcal{D}_{or}(L(u,K), L(v,K)).$$

To prove correctness, use induction on $k$. If $u$ and $v$ are neighbors of level $i$, then the edge $(u,v)$ appears in $F_i$ and therefore both $u$ and $v$ participate in either $G_{odd}$ or $G_{even}$ depending on the parity of $i$. Thus, by comparing the appropriate labels, say $L_{k-1}(G_{odd},u)$ and $L_{k-1}(G_{odd},v)$, we can deduce that $u$ and $v$ are indeed neighbors by the induction hypothesis.

If $u$ and $v$ are neighbors, $u$ is of level $i$, and $v$ is of level $i+1$, then the edge $(u,v)$ either appears in $F_i$ and $\mathsf{State}(v) = \mathsf{Dual}$ or appears in $K$ and $\mathsf{State}(v) = \mathsf{Even}$ or $\mathsf{Odd}$. Thus, if $(v,u)$ is in $F_i$ then, if $i$ is even, both vertices participate in $G_{even}$ and, if $i$ is odd, then both vertices participate in $G_{odd}$. By comparing the appropriate labels of $u$ and $v$ (either their $L(k-1, G_{even})$ label or their $L(k-1, G_{odd})$ label) and by using the induction hypothesis we are able to deduce that $u$ and $v$ are indeed neighbors.

If $\mathsf{State}(v) = \mathsf{Even}$ or $\mathsf{Odd}$, then the edge $(v,u)$ is in $K$, so by looking at $L(v,K)$ in $L_k(G,v)$ and detecting $id(u)$ appearing there, we conclude that $u$ and $v$ are indeed neighbors.

It is clear that if $u$ and $v$ are *not* neighbors in $G$, then they are not neighbors in either one of the decomposed subgraphs, and therefore, by the induction hypothesis we can never deduce that they are neighbors by our procedure. $\square$

We get the following corollary.

COROLLARY 4.19. *Let $\mathcal{G}_n$ be the family of $n$-vertex graphs. Then* $\mathcal{L}(k\text{–}\mathtt{v\text{-}conn}, \mathcal{G}_n) \leq 2^k \log n$.

**5. A lower bound for vertex-connectivity on general graphs.** In this section we establish a lower bound of $\Omega(k \log n)$ on the required label size for $k$-vertex-connectivity on the class of $n$-vertex graphs, where $k$ is polylogarithmic in $n$. Fix a constant integer $c \geq 1$, assume that $k \leq \log^c n$, and for $m = \frac{n}{2(k^2-k)}$ let $\mathcal{G}_m$ be the class of all $m$-vertex graphs $\langle V, E \rangle$ with fixed id's $\{v_1, \ldots, v_m\}$ and degree at most $k-1$. Transform a given graph $G \in \mathcal{G}_m$ into a graph $T(G) = H$ with $n$ vertices in the following way. Replace each edge $e_{i,j} = (v_i, v_j)$ in $G$ with $k$ vertices $w^1_{i,j}$ through $w^k_{i,j}$ and connect all the $w^l_{i,j}$'s to both $v_i$ and $v_j$. Since $G$ has at most $\frac{n}{2k}$ edges, $H$ has at most $n$ vertices. If necessary, add arbitrary isolated vertices to $H$ so that it has precisely $n$ vertices.

Observation 5.1. *Two vertices $v_i, v_j$ are adjacent in $G$ iff $u$ and $v$ are $k$-vertex-connected in $T(G) = H$.*

Assume we have a labeling scheme $\langle \mathcal{M}, \mathcal{D} \rangle$ for $k$-vertex connectivity on $n$-vertex graphs.

Observation 5.2. *Consider two distinct graphs $G_1, G_2 \in \mathcal{G}_m$, and let $L_i = \mathcal{M}(T(G_i))$ for $i = 1, 2$. Then there exists a vertex $v_j$ in $V$ such that $L_1(v_j) \neq L_2(v_j)$, i.e., $\{L_1(v_1), \ldots, L_1(v_m)\} \neq \{L_2(v_1), \ldots, L_2(v_m)\}$.*

Since the number of graphs in $\mathcal{G}_m$ is $(\frac{m}{k})^{\Omega(km)}$, which is $m^{\Omega(km)}$ for $k$ polylogarithmic in $n$, we get the following corollary.

Corollary 5.3. *There exists a graph $G \in \mathcal{G}(k)$ such that $\{L(v_1), \ldots, L(v_m)\}$ consists of at least $\log m^{\Omega(km)} = \Omega(km \log m)$ bits, where $L = \mathcal{M}(G)$.*

We get the following theorem.

Theorem 5.4. *$\mathcal{L}(k\text{−}\texttt{v-conn}, \mathcal{G}_n) = \Omega(k \log m) = \Omega(k \log n)$ for $k$ polylogarithmic in $n$.*

## REFERENCES

[1] S. Alstrup, P. Bille, and T. Rauhe, *Labeling schemes for small distances in trees*, in Proceedings of the 14th Annual ACM-SIAM Symposium on Discrete Algorithms, ACM, New York, SIAM, Philadelphia, 2003, pp. 689–698.

[2] S. Alstrup, C. Gavoille, H. Kaplan, and T. Rauhe, *Identifying Nearest Common Ancestors in a Distributed Environment*, Tech. report IT-C Series 2001-6, The IT University of Copenhagen, Denmark, August 2001.

[3] S. Abiteboul, H. Kaplan, and T. Milo, *Compact labeling schemes for ancestor queries*, in Proceedings of the 12th Annual ACM-SIAM Symposium on Discrete Algorithms, ACM, New York, SIAM, Philadelphia, 2001, pp. 547–556.

[4] M. A. Breuer and J. Folkman, *An unexpected result on coding the vertices of a graph*, J. Math. Anal. Appl., 20 (1967), pp. 583–600.

[5] M. A. Breuer, *Coding the vertexes of a graph*, IEEE Trans. Inform. Theory, IT-12 (1966), pp. 148–153.

[6] M. Chrobak and D. Eppstein, *Planar orientations with low out-degree and compaction of adjacency matrices*, Theoret. Comput. Sci., 86 (1991), pp. 243–266.

[7] S. Even, *Graph Algorithms*, Computer Science Press, Woodland Hills, CA, 1979.

[8] C. Gavoille, M. Katz, N. A. Katz, C. Paul, and D. Peleg, *Approximate distance labeling schemes*, in Proceedings of the 9th Annual European Symposium on Algorithms, Lecture Notes in Comput. Sci. 2161, F. Meyer auf der Heide, ed., Springer-Verlag, Berlin, 2001, pp. 476–488.

[9] C. Gavoille and C. Paul, *Distance labeling scheme and split decomposition,* Discrete Math., 273 (2003), pp. 115–130.

[10] C. Gavoille, D. Peleg, S. Pérennes, and R. Raz, *Distance labeling in graphs*, in Proceedings of the 12th Annual ACM-SIAM Symposium on Discrete Algorithms, ACM, New York, SIAM, Philadelphia, 2001, pp. 210–219.

[11] M. Katz, N. A. Katz, and D. Peleg, *Distance labeling schemes for well-separated graph classes*, in Proceedings of the 17th Symposium on Theoretical Aspects of Computer Science, Lecture Notes in Comput. Sci. 1770, Springer-Verlag, Berlin, 2000, pp. 516–528.

[12] H. Kaplan and T. Milo, *Parent and Ancestor Queries Using a Compact Index*, manuscript, 2001.

[13] H. Kaplan and T. Milo, *Short and simple labels for small distances and other functions*, in Proceedings of the Workshop on Algorithms and Data Structures, Lecture Notes in Comput. Sci. 2125, 2001, pp. 246–257.

[14] S. Kannan, M. Naor, and S. Rudich, *Implicit representation of graphs*, in Proceedings of the 20th ACM Symposium on Theory of Computing, ACM, New York, 1988, pp. 334–343.

[15] A. Korman and D. Peleg, *Labeling schemes for weighted dynamic trees*, in Proceedings of the 30th International Colloquim on Automata, Languages and Programming, Lecture Notes in Comput. Sci. 2719, Springer-Verlag, Eindhoven, The Netherlands, 2003, pp. 369–383.

[16] A. Korman, D. Peleg, and Y. Rodeh, *Labeling schemes for dynamic tree networks*, Theory Comput. Syst., 37 (2004), pp. 49–75.

[17] D. Peleg, *Proximity-preserving labeling schemes and their applications*, in Proceedings of the 25th International Workshop on Graph-Theoretic Concepts in Computer Science, Lecture Notes in Comput. Sci. 1665, Springer-Verlag, Berlin, 1999, pp. 30–41.

[18] D. Peleg, *Informative labeling schemes for graphs*, in Proceedings of the 25th Symposium on Mathematical Foundations of Computer Science, Lecture Notes in Comput. Sci. 1893, Springer-Verlag, Berlin, New York, 2000, pp. 579–588.

SIAM J. Comput.
Vol. 34, No. 1, pp. 41–66

# A CHARACTERIZATION OF UNIVERSAL STABILITY IN THE ADVERSARIAL QUEUING MODEL[*]

CARME ÀLVAREZ[†], MARIA BLESA[†], AND MARIA SERNA[†]

**Abstract.** We study universal stability of directed and undirected graphs in the adversarial queuing model for static packet routing. In this setting, packets are injected in some edge and have to traverse a predefined path before leaving the system. Restrictions on the allowed packet trajectory provide a way to analyze stability under different packet trajectories. We consider five packet trajectories, two for directed graphs and three for undirected graphs, and provide polynomial time algorithms for testing universal stability when considering each of them. In each case we obtain a different characterization of the universal stability property in terms of a set of forbidden subgraphs. Thus we show that variations of the allowed packet trajectory lead to nonequivalent characterizations.

Using those characterizations we are also able to provide polynomial time algorithms for testing stability under the NTG-LIS (Nearest To Go-Longest In System) protocol.

**Key words.** interconnection networks, adversarial queueing theory, greedy scheduling protocols, network stability, graph algorithms

**AMS subject classifications.** 68Q25, 68R10, 90B1, 90B22

**DOI.** 10.1137/S0097539703435522

## 1. Introduction.

One of the main goals in the study of the behavior of packet-switched communication networks is to determine the conditions of *stability*, i.e., the fact that the number of packets in the system remains bounded as the system dynamically evolves in time. Stability is studied considering that a communication system is formed by three main components: the network, the scheduling protocol, and the traffic pattern. *Networks* are modeled with (directed or undirected) graphs in which nodes represent the hosts and edges represent the links between these hosts. The *protocol* is used to schedule a packet when more than one packet wants to cross the same edge at the same time step. Packets waiting to traverse an edge are kept in a queue at the tail of the edge. The protocol determines the order in which the waiting packets cross the edge. The *traffic pattern* controls where and how packets join the system and defines their trajectory.

*Adversarial models.* The problem of deciding stability has been investigated under various models of packet routing [8, 9, 7, 12, 3]. Some models make probabilistic assumptions on the traffic pattern, while others replace more traditional stochastic arrival assumptions by worst-case inputs in the traffic pattern to perform a worst-case analysis. These latter models are closer to the traditional analysis of algorithms and to real network configurations. The *adversarial queuing theory* proposed by Borodin et al. [6], which is a robust model of queuing theory in network traffic, can be considered as the pioneering work in studying stability via worst-case analysis.

[*]Received by the editors September 29, 2003; accepted for publication (in revised form) April 9, 2004; published electronically October 1, 2004. This work was partially supported by the FET Programme of the EU under contracts IST-1999-14186 (ALCOM-FT) and IST-2001-33116 (FLAGS), and by the Spanish CICYT projects TIC-2001-4917-E and TIC-2002-04498-C05-03 (TRACER). The second author was also supported by the Catalan Research Council of the Generalitat de Catalunya under grant 2001FI-00659. Part of this work was presented in *Universal stability of undirected graphs in the adversarial queueing model*, Proceedings of the 14th ACM Symposium on Parallel Algorithms and Architectures (SPAA'02), Winnipeg, MB Canada, 2002.

http://www.siam.org/journals/sicomp/34-1/43552.html

[†]Departamento de Llenguatges i Sistemes Informàtics, Universitat Politècnica Catalunya, Campus Nord, E-08034 Barcelona, Spain (alvarez@lsi.upc.es, mjblesa@lsi.upc.es, mjserna@lsi.upc.es).

Adversarial models consider the time evolution of a packet-routing network as a game between an adversary and a protocol. The adversary may inject a set of packets at some nodes at each time step. In this work, we consider static packet routing; in this setting, the adversary also specifies for each packet the complete path that the packet must traverse.

Adversarial models have been shown to be good theoretical frameworks for traffic patterns in modern communication networks. These models can reflect the behavior of connection-oriented networks with transient connections (such as ATM networks) as well as connectionless networks (such as the Internet). Different factors are used in those adversarial models to describe the adversary and quantify its power. Three of these factors refer to the *injection rate* (i.e., the frequency at which the adversary introduces packets into the network), the *burstiness* (i.e., the maximum number of packets that can be injected in an edge in one step), and the *initial configuration* (i.e., the initial quantity of packets distributed over the network at time zero). Restrictions on the quantity of packets injected are introduced in order to avoid adversaries that could trivially collapse the system. In general, during each interval of time (that can be defined in different ways), the number of packets injected during that time interval requiring any edge in their trajectory cannot exceed a certain bound proportional to the length of the time interval. Two main adversarial models have been considered in the last years, both of them assuming an empty initial configuration:

*The windowed adversarial (queuing) model* by Borodin et al. [6]. An adversary in this model is restricted by two parameters $(w, r)$, where $w \geq 1$ is the window size and $0 < r \leq 1$ is the injection rate. The adversary is allowed to inject sequences of packets under the restriction that, at any $w$ consecutive time steps, the total number of packets requiring any concrete edge $e$ is at most $\lceil rw \rceil$.

*The leaky-bucket adversarial model* by Andrews et al. [4]. An adversary in this model is also restricted by two parameters $(b, r)$, where $b \geq 0$ is the burstiness and $0 < r \leq 1$ is the injection rate. The adversary is allowed to inject sequences of packets under the load condition that, of the packets that the adversary injects in any interval $T$, at most $\lceil r|T| \rceil + b$ may have paths that contain the same particular edge.

In a recent work Rosén compares the relative power of the *windowed* and the *leaky-bucket* adversarial models [18]. For injection rates $r < 1$, adversaries in one model can be simulated by adversaries in the other model injecting the same sequence of packets. Thus, the results for one model also hold for the other model. However, when $r \leq 1$ a leaky-bucket adversary is more powerful than a windowed adversary, since there are some sequences of packets that can be produced by the former when $r = 1$, but cannot be produced by the latter.

In this work we will study universal stability of directed and undirected graphs when $r < 1$ under the leaky-bucket adversarial model, as it is done in [4]. In general, stability results are shown for any network with an empty initial configuration, while instability results are shown starting from a network with a given nonempty initial configuration (see [6] or [4]). The results in [4] show that any instability result for a network with a given initial configuration can be translated into an instability result with empty initial configuration for a different network. These results can be used to show the nonuniversal stability of a given protocol but not to characterize universal stability of networks. However, we will show that network stability is independent of whether the initial configuration is empty or not.

*Greedy protocols* are those that forward a packet across an edge $e$ whenever there is at least one packet stored in the queue of $e$ waiting to traverse the edge. Some natural greedy protocols are FIFO (First In First Out), in which highest priority is given to the packet that has arrived first in the queue; LIS (Longest In System), in which every queue gives priority to the packet that has been in the system the longest and; and NTG (Nearest To Go), in which highest priority is assigned to the packet that still has to cross the smallest number of edges. Other protocols are FTG (Furthest To Go), NTS (Nearest To Source), SIS (Shortest In System), LIFO (Last In First Out), and FFS (Farthest From Source).

In this paper, as customary in the literature about stability, we consider only greedy protocols. All the instability results in this paper will be shown under the NTG-LIS protocol, which works as the NTG protocol and solves ties using LIS.

*The packet trajectory* refers to the form of the paths that packets are allowed to follow over the network. For directed graphs, two different types of paths have been considered in the literature. Borodin et al. [6] and Andrews et al. [4] assumed that when a packet is injected, its assigned path does not contain any edge more than once (see page 45 of [4]); however, a different definition of packet trajectory was considered by Goel in [14]. Although a detailed definition is not given in [14], only paths that do not contain any vertex more than once are allowed (see, for example, the comment just before Lemma 2.4 on page 221 of [14]). Hence, the paths considered by Goel (referred to in the following as *simple paths*) are a restrictive version of the paths considered by Andrews et al. (referred to in the following as *paths*).

In the case of undirected graphs, Andrews et al. assume that packets can use only paths that do not cross the same edge twice. This means that a packet cannot traverse the same edge in both directions (see page 57 of [4]). We refer to these paths as *undirected paths.* Together with this type of path, we will also consider *paths* and *simple paths* defined over the directed version of the undirected graph.

Gamarnik [13] considers undirected graphs and uses a different model for the allowed packet traffic: each edge is undirected and can carry only a single packet in one step. This contrasts with the Andrews et al. model in which the edges can be seen as bidirected, and each edge can carry a packet in each direction at each step.

*Known results.* A network is said to be universally stable when the systems composed by that concrete network are stable regardless of the selected protocol and traffic pattern. A protocol is said to be universally stable when all the systems that use it are stable. The existence of networks and scheduling protocols that are (respectively, are not) universally stable was initially shown in [6] and [4].

Until the work of Rosén [18], it was not clear if both the windowed and the leaky-bucket models were equivalent. Since it was shown that they are for injection rates $r < 1$, results for one model also hold for the other model when the injection ratio accomplishes this condition. Keeping this equivalence in mind, let us summarize some of the most important results obtained in the respective models.

In the windowed adversarial model, universal stability of networks with tree, mesh, and directed acyclic topologies was shown to hold in [6]. On ring topologies, protocols LIS and FIFO were shown to be nonstable with injection rate $r = 1$, whereas FTG was shown to be stable. Concerning only protocols, it was also shown in [6] that, for every $r > 0$, there exists a queuing network for which NTG is nonstable at rate $r$. Every greedy protocol is shown in [16] to be stable if the injection rate is not more than $1/(d + 1)$, where $d$ denotes the diameter of the network.[1] Much effort has been

[1]The diameter of a graph is the length of the longest path in it that does not pass twice over the same edge.

dedicated to the FIFO protocol, for which the best known lower bound for instability was improved in [16] down to 0.5.

In the leaky-bucket adversarial model with $r < 1$, stability of networks describing a ring topology was shown to hold under any greedy protocol (see [4]). In the same work, some commonly used simple greedy protocols (namely, FTG, NTS, SIS, and LIS) were shown to be universally stable, while some others (namely, FIFO, LIFO, NTG, and FFS) were shown not to be universally stable. Considering that the system might have initial configuration, stability and instability properties of FIFO have been recently studied in this model. A network-dependent constant is provided in [10] such that FIFO is stable against any adversary with a smaller injection rate. A lower bound of 0.749 for the instability of FIFO was given in [15], where the stability of networks with heterogeneous protocols also was addressed. Moreover, in [19] it was shown that FIFO is stable when the injection rate is smaller than $1/(d-1)$. In this model, FIFO has been shown to be unstable at arbitrarily low injection rates [5].

With regard to the universal stability of networks one of the first questions that arose was whether it would be possible to detect stability from the knowledge of the topological structure of the network and the scheduling protocols. For undirected graphs, Andrews et al. show in [4] that, for a particular type of packet trajectory (the one we call *undirected path*), cycles and trees are universally stable. Furthermore, they also show that the family of *undirected-path universally stable* graphs is minor-closed and that there exists a finite set of basic undirected graphs such that a graph is stable if and only if it does not contain as a minor any of the graphs in that set. These results guarantee decidability in polynomial time; however, a constructive proof is not presented.

*Our results.* In this paper we consider the computational complexity of deciding universal stability of directed and undirected graphs. We consider different restrictions on the type of path that the packets can follow, i.e., the packet trajectory. We would like to highlight the importance of specifying the type of path since, for each category, the characterization of universal stability is different. For each considered case we obtain a characterization in terms of forbidden subgraphs and provide an explicit polynomial time algorithm for deciding the property of stability. Concerning directed graphs, and under the assumption that packets follow simple paths, we obtain a characterization of universal stability that disproves the characterization in [14] under the same assumption (which was presented in terms of the forbidden minors given in Figure 1). Further comments concerning this fact will be given after presenting our characterization.

An interesting question concerning stability is that of deciding the stability of a concrete network under a fixed protocol. Using the fact that all the instability results obtained in this paper apply to networks under the NTG-LIS protocol, we can show that the problem of deciding whether a network is stable under the NTG-LIS protocol can also be solved in polynomial time.

*Organization.* Section 2 sets out the definitions of all models considered in this work and some preliminary results. In section 3, universal stability of some particular directed and undirected graphs is shown. Section 4 contains the basic results on instability of directed and undirected graphs. Finally, section 5 details the characterizations for universal stability in terms of forbidden subgraphs, and section 6 provides alternative characterizations in terms of graph properties together with the algorithms that check them. Finally, section 7 shows the polynomial time decidability of checking for stability under the NTG-LIS protocol. For sake of readability several technical proofs have been delayed to the appendix.

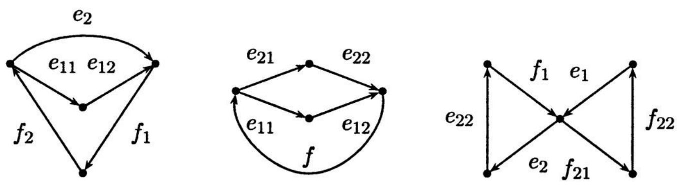


FIG. 1. *Forbidden minors proposed in* [14] *for simple-path universal stability of digraphs.*

## 2. Preliminaries.

**2. Preliminaries.** All the graphs in this paper may have multiple edges but no loops. Multiple edges share the same pair of different endpoints, while the endpoints of a loop are the same vertex. We will use the term *digraph* to refer to a directed graph and simply *graph* for an undirected graph. For a digraph we will use the term *arc* instead of edge. Given a graph $G$, $G^d$ denotes the digraph formed from the same vertex set as $G$ but replacing every edge $\{u, v\}$ in $G$ with the two arcs $(u, v)$ and $(v, u)$.

Two edges are *incident* if they share at least one vertex. A *walk* is an alternating sequence of vertices and edges (respectively, arcs) in the form

$$v_0, e_1, v_1, \dots, v_{n-1}, e_n, v_n,$$

where for each $i$, $1 \leq i \leq n$, $e_i = \{v_{i-1}, v_i\}$ (respectively, $e_i = (v_{i-1}, v_i)$) and each $v_j$ is a vertex. A *path* is a walk in which all the edges (respectively, arcs) are different. A *simple path* is a walk in which all the vertices, and thus necessarily all the edges (respectively, arcs) are different. A walk is *closed* if $v_0 = v_n$. A closed walk is an $n$-*cycle* provided its $n$ vertices are distinct.

In this work we adopt, without loss of generality, the leaky-bucket definition of adversary which is defined by two parameters $(r, b)$, where $b \geq 0$ and $0 < r \leq 1$. An $(r, b)$-*adversary* (or just an *adversary*) is allowed to inject sequences of packets under the load condition that, of the packets that the adversary injects in any interval $T$, at most $\lceil r|T| \rceil + b$ packets may have trajectories that contain any particular edge. Rosén [18] showed that adversaries in the windowed and in the leaky-bucket models (starting with an empty configuration) have the same power provided that $r < 1$. The equivalence requires only a change in the parameters of the adversary, not in the sequence of packet trajectories; therefore it provides a valid equivalence for all our subclasses of adversaries.

When in addition to an $(r, b)$-adversary $\mathcal{A}$, we are given an initial configuration $C$, we can define a new $(r, b')$-adversary $\mathcal{A}$' as follows:

Let $b' = b + |C|$; then $\mathcal{A}'$ injects all the packets in $C$ at time step 1, and at any other time step $t > 1$, the same set of packets that $\mathcal{A}$ would inject at time $t - 1$.

Here the initial configuration means the set of packets that are in the system initially. Both systems, with empty or nonempty initial configuration, behave alike; hence we can work equivalently with empty or not empty initial configuration. Since only the parameters of the adversary have to be changed, this remark is also valid for any system with some restrictions on the packet trajectory. Notice that this is a stronger result than the analogous result given in [4] since here the graph does not need to be changed (if the graph topology is changed, then results on stability of networks cannot be translated from one model to the other).

Throughout the paper we will use the leaky-bucket adversarial model with empty

Table 1
*Summary of concepts.*

| Directed Graphs | | |
|---|---|---|
| Packet trajectory | Characteristics of the route | Stability term |
| path | walk with nonrepeated edges | stability |
| simple path | walk with nonrepeated vertices | simple-path stability |

| Undirected Graphs | | |
|---|---|---|
| Packet trajectory | Characteristics of the route | Stability term |
| path | path in $G^d$ | stability |
| undirected path | path in $G$ | undirected-path stability |
| simple path | simple path in $G^d$ | simple-path stability |

initial configuration and $r < 1$. However, for the sake of simplicity, we will use nonempty initial configurations when describing adversaries causing network instability. In the following, we establish the definitions of stability in the adversarial queuing model used in this work. The different definitions are summarized in Table 1.

Definition 1. *Given a network $G$, a protocol $P$, and an adversary $\mathcal{A}$, we say that the system $(G, P, \mathcal{A})$ is* stable *if the number of packets in the system is always bounded.*

**2.1. Networks as digraphs.** When the network is represented by a digraph $G$, we consider two classes of packet trajectories, thus giving rise to two adversary classes: an *adversary* can use as packet trajectory any path in $G$, while a *simple-path adversary* can use as packet trajectory only simple paths in $G$. Accordingly we set the two definitions of stability under a protocol.

Definition 2. *Given a digraph $G$ and a protocol $P$, we say that*

– *$G$ is* stable *under protocol $P$ (the pair $(G, P)$ is stable) if for any adversary $\mathcal{A}$, the system $(G, P, \mathcal{A})$ is stable.*
– *$G$ is* simple-path stable *under protocol $P$ (the pair $(G, P)$ is simple-path stable) if for any simple-path adversary $\mathcal{A}$, the system $(G, P, \mathcal{A})$ is stable.*

Similarly, we define universal stability of digraphs in the following form.

Definition 3. *A digraph $G$ is* universally stable *if, for any protocol $P$, the pair $(G, P)$ is* stable*. A digraph $G$ is* simple-path universally stable *if, for any protocol $P$, the pair $(G, P)$ is* simple-path stable.

Observe that any universally stable digraph is also simple-path universally stable, but the opposite, as we will see, is not true.

The property of simple-path universal stability was shown in [14] to be maintained when acyclically joining simple-path universally stable digraphs. A closer inspection of the proof reveals its validity for the two proposed models for directed graphs proposed in this work.

Lemma 1. *If digraphs $G_1$ and $G_2$ are (simple-path) universally stable, then so is any graph $G$ formed by joining them with edges that go only from $G_1$ to $G_2$.*

As a consequence of the previous result we have the following theorem.

Theorem 1. *A digraph $G$ is (simple-path) universally stable if and only if all its strongly connected components are (simple-path) universally stable.*

**2.2. Networks as graphs.** When representing networks by undirected graphs, we consider three different packet trajectory restrictions: given a graph $G$, an *adversary* can use as packet trajectory any path in $G^d$; an *undirected-path adversary* can use as packet trajectory any path in $G$; and a *simple-path adversary* can use as packet trajectory only simple paths in $G^d$.

Observe that, in the first case, an edge can be used twice by the same packet provided it is traversed in opposite directions, but both directions have different queues and thus this model is different from Gamarnik's proposal [13]. In the second model the same edge can be traversed only once. This corresponds with the model used by Andrews et al. in [4]. The third model does not allow a packet to pass twice through the same vertex. Notice that, in this latter model, a multiedge can be traversed only once and only in one direction. Observe that the condition *simple-path in G* is equivalent to *simple-path in $G^d$*, so the fourth possible model has already been considered.

As before, we set the definitions of stability under a protocol.

Definition 4. *Given a graph $G$ and a protocol $P$, we say that*

- *$G$ is* stable *under protocol $P$ (the pair $(G,P)$ is stable) if for any adversary $\mathcal{A}$, the system $(G,P,\mathcal{A})$ is stable.*
- *$G$ is* undirected-path stable *under protocol $P$ (the pair $(G,P)$ is undirected-path stable) if for any undirected-path adversary $\mathcal{A}$, the system $(G,P,\mathcal{A})$ is stable.*
- *$G$ is* simple-path stable *under protocol $P$ (the pair $(G,P)$ is simple-path stable) if for any simple-path adversary $\mathcal{A}$, the system $(G,P,\mathcal{A})$ is stable.*

It is straightforward to show that if a pair $(G,P)$ is stable, then $(G,P)$ is also undirected-path stable, and if $(G,P)$ is undirected-path stable, then $(G,P)$ is simple-path stable. As we will see later, these inclusions are strict. Now we can write the corresponding definitions of universal stability.

Definition 5. *A graph $G$ is* universally stable *if for any protocol $P$ the pair $(G,P)$ is* stable*. A graph $G$ is* undirected-path universally stable *if for any protocol $P$ the pair $(G,P)$ is* undirected-path stable*. A graph $G$ is* simple-path universally stable *if for any protocol $P$ the pair $(G,P)$ is* simple-path stable*.*

The universal stability of graphs under the undirected-path model (as defined above) was addressed in [4] where it is proved to be closed under minors and therefore decidable in polynomial time; however, no constructive algorithm is known. For the other models this question was unresolved. We show a constructive way of deciding the universal stability property also for the other models.

We can also state an equivalent result to that in Theorem 1 for the three forms of universal stability for undirected graphs presented here. Note that in the case of undirected graphs, each pair of connected components are independent, i.e., there is no edge connecting them.

Theorem 2. *A graph $G$ is (simple-path $|$ undirected-path) universally stable if and only if all its connected components are (simple-path $|$ undirected-path) universally stable.*

Finally, let us remark that in any of the five packet trajectory models considered in this paper (i.e., two for digraphs and three for graphs), the instability of a sub(di)graph implies the instability of the whole (di)graph.

## 3. Some universally stable graphs and digraphs.

In the following, we use standard graph terminology to denote the following graphs and digraphs: directed and undirected *trees*, the *cycle* on $k$ vertices ($k \geq 2$), the *directed cycle* on $k$ vertices ($k \geq 2$), and *directed acyclic* graphs. For graphs and digraphs with multiple edges we define

- a *unicyclic graph* as those graphs that contain only one cycle (see Figure 2(b.3));

– a *multi-tree* as an undirected tree with multiple edges (see Figure 2(c.1));
– a *decorated cycle* as being obtained from a $k$-cycle with $k \geq 3$, and some multitrees (see Figure 2(c.2)), after identifying one vertex from each tree with a vertex from the cycle;
– an *oriented multitree* as the directed version of an undirected tree in which each edge is substituted by two arcs (one in each direction), and which can also contain multiple arcs (see Figure 2(c.3)). Note that all the simple directed cycles have two vertices;
– a *decorated directed cycle* as the directed version of a $k$-cycle with $k \geq 3$, and some oriented multitrees, after identifying one vertex from each "tree" with a vertex from the cycle (see Figure 2(c.4)). Note that the obtained graph is strongly connected.

In this section we prove the universal stability of some graphs and digraphs according to the different packet trajectories considered. Previous results from [6] and [4] are rewritten as Lemmas 2 and 3 according to the terminology introduced in this work.

LEMMA 2. *All acyclic digraphs and the directed cycle on any number of vertices are universally stable.*

As we have commented before, universal stability of digraphs implies simple-path universal stability; therefore, acyclic digraphs and directed cycles are also simple-path universally stable. Concerning graphs, we know that the following lemmas hold.

LEMMA 3. *All trees and cycles on any number of vertices are undirected-path universally stable.*

Now we give another family of undirected-path universally stable graphs, thus extending the previous result.

LEMMA 4. *All unicyclic graphs are undirected-path universally stable.*

*Proof.* Observe that, if $G$ is a unicyclic graph, removing the edges in the cycle results in a forest. We root each tree in this forest at the vertex that is common with the cycle; this gives an orientation *upwards* or *downwards* to every arc in $G^d$.

We classify the packets into three flow types. Flow type $\alpha$ is formed by those packets injected in an upward arc. Flow type $\beta$ is formed by packets injected in a cycle arc; in fact, we will consider this flow split into two flows, depending on whether the initial arc follows one of the two cycle orientations. Finally, flow type $\gamma$ is formed by the packets injected in a downward arc.

Observe that packets starting in a downward arc can follow only downward arcs; otherwise they will use twice an edge in $G$. Similarly, packets starting in the cycle cannot change the initial cycle orientation. Therefore they either die in the cycle or leave the cycle using a downward arc. This provides a directed acyclic interaction of the three flow types. Upward edges can carry only packets from $\alpha$ flow. As directed acyclic graphs are undirected-path stable, the corresponding queues will have bounded maximum size. The edges in the cycle can be considered as two directed cycles, as no flow can be passed from one to the other. These cycle edges can get packets from $\alpha$ flow and $\beta$ flow. In this situation we have a stable network (upward edges) entering in another stable network (one of the directed cycles) and, by Lemma 1, the corresponding queues will have bounded maximum size. For the downward arcs we have the same situation, a flow coming from a stable network entering in another stable network. Thus the result follows. $\square$

All the families of graphs which are undirected-path universally stable are also simple-path universally stable. However, we can enlarge the set of simple-path universally stable graphs by allowing multiple edges.

LEMMA 5. *A multitree is simple-path universally stable.*

*Proof.* Observe that a packet can use only once, and only in one direction, one of the multiple edges connecting the same pair of vertices. The argument is similar to the one in Lemma 4: we root the tree in order to assign an upward/downward direction to the edges in $G^d$. The simple-path requirement prevents packets starting in a downward arc from using an upward arc (even in the case where the arcs in $G^d$ come from different edges in $G$), because in this case, a vertex would be visited twice. Thus, we can conclude the simple-path universal stability of a multitree. □

LEMMA 6. *A decorated cycle is simple-path universally stable.*

*Proof.* The result follows from Lemmas 4 and 5 using the same arguments as in Lemma 5. □

LEMMA 7. *A oriented multitree and a decorated directed cycle are simple-path universally stable.*

*Proof.* The result follows from Lemmas 5 and 7 using the arguments similar to those in Lemmas 4 and 5. □

Figure 2 summarizes and classifies the families of graphs and digraphs that are universally stable according to the different packet trajectories considered.

## 4. Some graphs and digraphs that are not universally stable.

Once the families of universally stable graphs and digraphs are identified for each of the cases considered, we focus on detecting which are the simplest graphs and digraphs that are not stable. By iteratively applying subdivision operations to those simplest graphs and digraphs we will "extend" them and we will define families of graphs. Each family will characterize one of the cases of universal stability introduced in this work. We consider the following subdivision operations on graphs and digraphs:

- The *subdivision of an edge* $\{u, v\}$ in a graph $G$ consists of the addition of a new vertex $w$ and the replacement of $\{u, v\}$ by the two edges $\{u, w\}$ and $\{w, v\}$.
- The *subdivision of an arc* $(u, v)$ in a digraph $G$ consists of the addition of a new vertex $w$ and the replacement of $(u, v)$ by the two arcs $(u, w)$ and $(w, v)$.
- The *subdivision of a 2-cycle* $(u, v)$, $(v, u)$ in a digraph $G$ consists of the addition of a new vertex $w$ and the replacement of $(u, v)$, $(v, u)$ by the arcs $(u, w)$, $(w, u)$, $(v, w)$, and $(w, v)$.

Given a graph $G$, $\mathcal{E}(G)$ denotes the family of graphs formed by $G$ and all the graphs obtained from $G$ by successive edge subdivisions. Given a digraph $G$, $\mathcal{E}(G)$ denotes the family of digraphs formed by $G$ and all the digraphs obtained from $G$ by successive arc or 2-cycle subdivisions. Observe that, for a graph $G$, $\mathcal{E}(G)^d \subseteq \mathcal{E}(G^d)$, but it might be the case that $\mathcal{E}(G)^d \neq \mathcal{E}(G^d)$; see Figure 3 for an example.

In this section we will prove instability results of networks under the NTG-LIS protocol. To simplify the notation, a path is specified by the sequence of its edges. First we show that some simple graphs are not stable and second we apply subdivision operations to these graphs and also show their instability. Observe that, as we are using the NTG protocol, if the length of the path that a packet has to traverse is increased, then its priority at a given edge can be changed. Therefore, edges composing a packet path cannot be replaced indistinctly with paths. However, for the particular graphs and adversaries we will deal with, the adversary can be adapted to provide an instability proof. The names used to denote the graphs correspond to the ones depicted in Figures 4 and 5.

THEOREM 3. *All the digraphs in* $\mathcal{E}(U_1) \cup \mathcal{E}(U_2)$ *are not stable under the* NTG-LIS *protocol.*

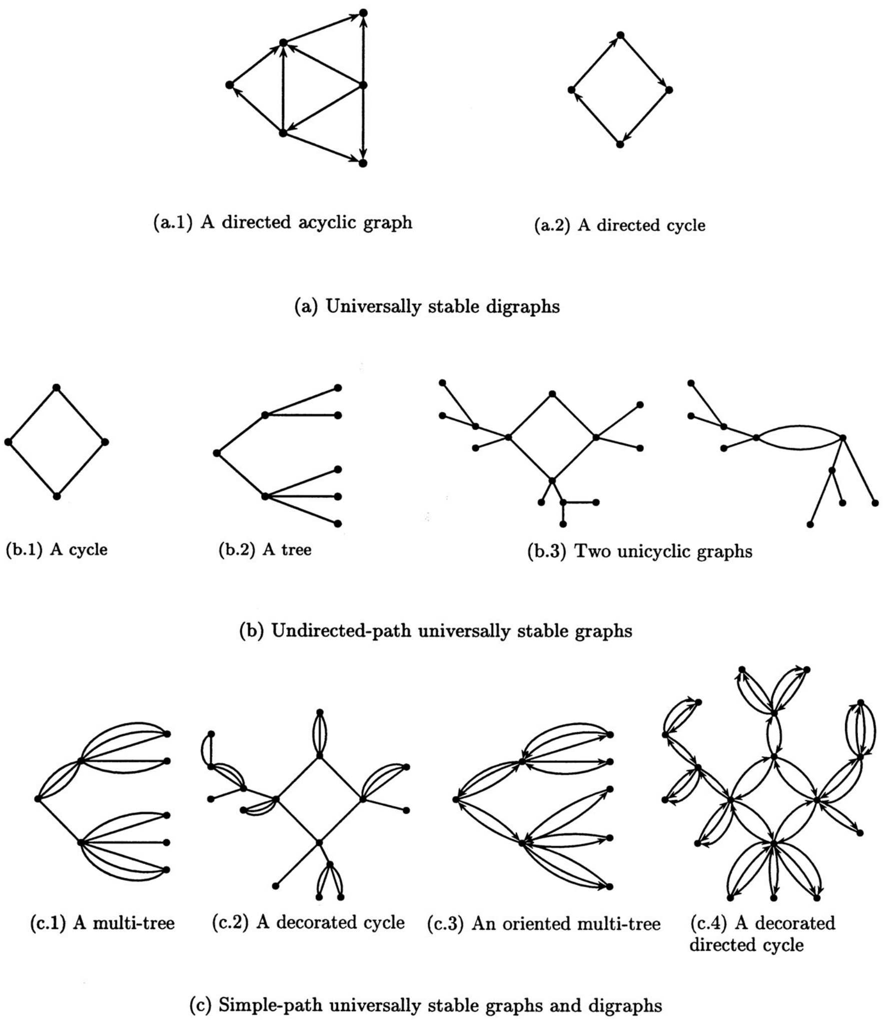

(a.1) A directed acyclic graph
(a.2) A directed cycle
(a) Universally stable digraphs
(b.1) A cycle
(b.2) A tree
(b.3) Two unicyclic graphs
(b) Undirected-path universally stable graphs
(c.1) A multi-tree
(c.2) A decorated cycle
(c.3) An oriented multi-tree
(c.4) A decorated directed cycle
(c) Simple-path universally stable graphs and digraphs

FIG. 2. *Examples of representatives of the families of universally stable graphs and digraphs in different cases considered in this work.*

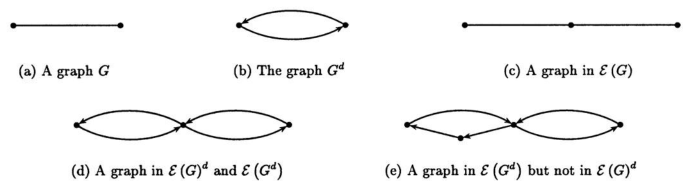
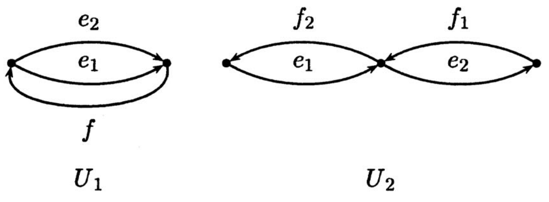
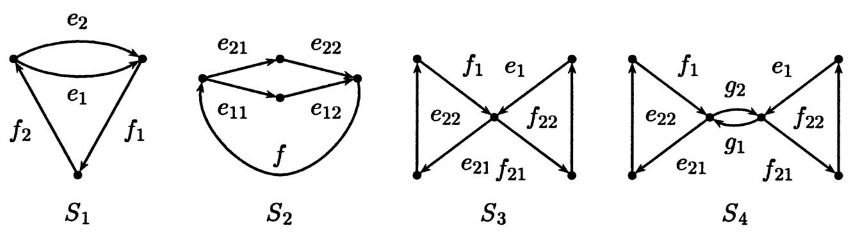

(a) A graph $G$
(b) The graph $G^d$
(c) A graph in $\mathcal{E}(G)$
(d) A graph in $\mathcal{E}(G)^d$ and $\mathcal{E}(G^d)$
(e) A graph in $\mathcal{E}(G^d)$ but not in $\mathcal{E}(G)^d$

FIG. 3. *An example illustrating the differences between* $\mathcal{E}(G)^d$ *and* $\mathcal{E}(G^d)$.

*Proof.* We sketch here the main lines and refer the reader to Appendix A where some of the most technical auxiliary results are proved. We start by showing that the pair $(U_1, \text{NTG-LIS})$ is not stable; to do so we provide an adversary and an initial configuration. The adversary works in rounds, and at the end of the presented rounds the network has a configuration with the same type of packets as in the initial con-

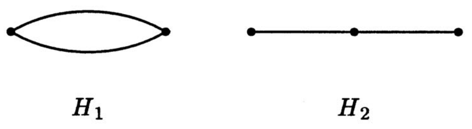


**(a) not universally stable digraphs**

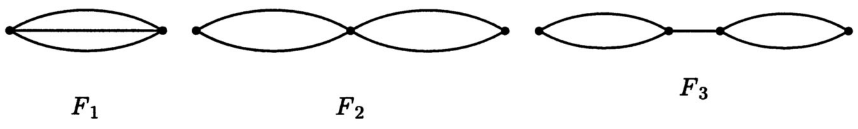


**(b) not simple-path universally stable digraphs**

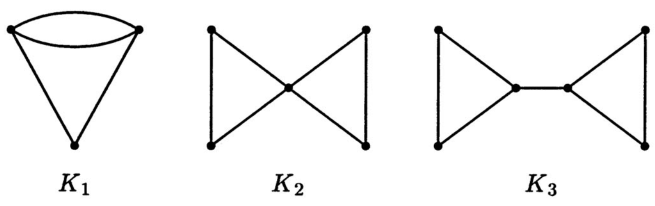
FIG. 4. *Some not universally stable digraphs.*



**(a) not universally stable graphs**



**(b) not undirected-path universally stable graphs**



**(c) not simple-path universally stable graphs**

FIG. 5. *Some not universally stable graphs.*

figuration but with an increased number of them. By repeatedly playing the set of rounds the system shows instability.

*Initial configuration.* At the beginning there are $s$ packets that want to traverse edge $f$; half of them are of the form $(e_1\ f)$, and half of them are of the form $(e_2\ f)$. The adversary $\mathcal{A}$ will play injections in four rounds.

*Round* 1. For $s$ steps, the adversary injects $rs$ packets of the form $(f\ e_2)$. These injections get mixed with the initial packets at edge $f$ and are blocked there because the queuing protocol is NTG.

*Round* 2. For the next $rs$ steps, the adversary injects a set of $r^2s$ packets of the form $(f\ e_1)$ and $r^2s$ packets of the form $(e_2)$. Injections $(f\ e_1)$ are blocked by the packets $(f\ e_2)$ from the first round because they are at the same distance to their destination and the secondary protocol that we apply is LIS. The $r^2s$ injections of the form $(e_2)$ are also blocked.

*Round* 3. For the next $r^2s$ steps, the adversary injects $r^3s$ packets of the form $(e_2)$ and $r^3s$ of the form $(e_1\ f)$. The simple injections on $e_2$ will be blocked by the packets at $e_2$ from the previous round. The $(e_1\ f)$ injections get mixed with the packets of the form $(f\ e_1)$ from the previous round at edge $f$, where the injections get blocked because their distance to destination is longer.

*Round* 4. For the next $r^3s$ steps, the adversary injects $r^4s$ packets of the form $(e_2\ f)$ and $r^4s$ packets of the form $(e_1)$. The simple injections at $e_1$ block the packets $(e_1\ f)$ from the previous round because their distance to destination is shorter. The $(e_2\ f)$ injections are blocked by the $(e_2)$ packets from the previous round.

At the end there are $2r^4s$ packets queued at $e_1$ and $e_2$ that want to traverse edge $f$, $r^4s$ are of the form $(e_1\ f)$, and $r^4s$ are of the form $(e_2\ f)$. If cycles are allowed, the adversary $\mathcal{A}$ described above makes the network $U_1$ nonstable when $2r^4s > s$, i.e., at injection rate $r > 0.84089$.

The second step is to prove instability for the pair $(U_2, \text{NTG-LIS})$, as is done in Lemma 9 of Appendix A. Once the two base digraphs are shown not to be stable, we have to show instability for any extension of them. Observe that only the extensions obtained by arc subdivision have to be considered since the extensions obtained by 2-cycle subdivision already contain $U_2$ as a subgraph. Therefore, we have only to consider the two extensions depicted in Figures 6(a) and 6(b) whose corresponding instability results are given in Lemmas 10 and 11 (see Appendix A). ∎

We remark that, for digraphs, simple-path stability and stability are not equivalent. The pair $(U_1,\text{NTG-LIS})$ is not stable although $U_1$ is simple-path universally stable. The last result is easy to obtain as the set of simple-paths in $U_1$ is the set $\{(e_1), (e_2), (f)\}$, and any adversary using this set of disjoint packet trajectories is equivalent to an adversary playing on a digraph with three isolated arcs. The latter digraph is acyclic and therefore universally stable. Similar reasoning can be applied to $U_2$. These considerations lead to digraphs $S_1$, $S_2$, $S_3$, and $S_4$ (see Figure 4(b)) as the smallest digraphs which are not simple-path universally stable.

THEOREM 4. *All the digraphs in $\mathcal{E}\,(S_1) \cup \mathcal{E}\,(S_2) \cup \mathcal{E}\,(S_3) \cup \mathcal{E}\,(S_4)$ are not simple-path stable under the* NTG-LIS *protocol.*

*Proof.* The proof follows the same lines as for Theorem 3. We first prove that $S_1$, $S_2$, $S_3$, and $S_4$ are not simple-path stable (Lemmas 12, 13, 14, and 15). The restriction on considering simple-path trajectories justifies the need for having cycles of length at least 3 in a simple-path nonstable digraph. Extensions by 2-cycle subdivision are applicable only to $S_4$. Therefore, to finish the proof we show that the digraphs corresponding to the extensions depicted in Figures 7(a), 7(b), 7(c), and 7(d) are not simple-path stable (Lemmas 16, 17, 18, 19, and 11). See Appendix B for details. □

THEOREM 5. *All the graphs in $\mathcal{E}(H_1) \cup \mathcal{E}(H_2)$ are not stable under the* NTG-LIS *protocol.*

*Proof.* First observe that $H_1^d$ contains $U_1$ as a subgraph and that $H_2^d$ is $U_2$. Furthermore, any graph in $\mathcal{E}(H_1) \cup \mathcal{E}(H_2)$ can be oriented to contain, as a subgraph, a graph in $\mathcal{E}(U_1) \cup \mathcal{E}(U_2)$; therefore its instability under NTG-LIS follows from Theorem 3. □

Observe that the graph $H_1$ is undirected-path universally stable; again the set of possible undirected-path packet trajectories is formed by nonoverlapping paths with length 1. Therefore, undirected-path stability and stability are nonequivalent properties. Similar reasoning can be applied to $H_2$.

To characterize undirected-path stability we need to consider some larger graphs. Furthermore, instead of considering a fully directed version of the graph, we will need to fix an orientation to all the edges, with the exception of those that are graph separators, as those edges may have to be crossed in both directions.

THEOREM 6. *All the graphs in $\mathcal{E}(F_1) \cup \mathcal{E}(F_2) \cup \mathcal{E}(F_3)$ are not undirected-path stable under the* NTG-LIS *protocol.*

*Proof.* Let us prove first that the graphs $F_1$ and $F_2$ are NTG-LIS undirected-path nonstable. Giving an adequate orientation to the edges in $F_1$ and $F_2$, we get the digraphs $U_1$ and $U_2$ and as each arc corresponds to a different edge, the instability follows. Maintaining these orientations, any graph in $G \in \mathcal{E}(F_1) \cup \mathcal{E}(F_2)$ can be identified with a digraph in $\mathcal{E}(U_1) \cup \mathcal{E}(U_2)$ and therefore $G$ is not undirected-path stable under NTG-LIS.

For the graph $F_3$ we can fix an orientation for the edges forming the 2-cycles but we have to keep both directions, where possible, for the middle edge. Although no packet can use both, different packets may traverse the edge in different directions. Starting with the proof of NTG-LIS instability for the graph $U_2$ and working in a similar way as in Lemmas 15 and 19 it follows that $F_3$ and any graph in $\mathcal{E}(F_3)$ are not undirected-path stable under NTG-LIS. □

The graph $F_1$ is simple-path universally stable but not undirected-path stable; therefore simple-path stability differs from undirected-path stability. Cycles of length 3 or longer are needed to obtain simple-path nonstable graphs, thus obtaining $K_1$, $K_2$, and $K_3$ as the smallest graphs which are not simple-path stable.

THEOREM 7. *All the graphs in $\mathcal{E}(K_1) \cup \mathcal{E}(K_2) \cup \mathcal{E}(K_3)$ are not simple-path stable under* NTG-LIS.

*Proof.* Observe that we can give an orientation to the edges in $K_1$ and $K_2$ to obtain the digraphs $S_1$ and $S_3$ that are not simple-path stable under NTG-LIS. For the graph $K_3$ we can orient the two cycles and maintain the two opposite arcs for the middle edge, obtaining the graph $S_4$. Theorem 4 gives the simple-path instability under NTG-LIS for any graph in $\mathcal{E}(K_1) \cup \mathcal{E}(K_2) \cup \mathcal{E}(K_3)$. □

**5. Characterizing universal stability.** In this section we provide characterizations for the universal stability property of digraphs and graphs for each of the five proposed adversarial models. As before, the graph nomenclature corresponds to Figures 4 and 5.

We first show the characterization of the simple-path universal stability property since, in our opinion, the properties defining the simple-path stability for graphs are easier to understand once the corresponding properties for simple-path stability of digraphs are given.

THEOREM 8. *A digraph is universally stable if and only if it does not contain as subgraphs any of the digraphs in* $\mathcal{E}(U_1) \cup \mathcal{E}(U_2)$.

*Proof.* The "only if" part follows from Theorem 3 and the fact that the instability of a subgraph implies the instability of the whole graph. If $G$ does not contain as a subgraph a digraph in $\mathcal{E}(U_1) \cup \mathcal{E}(U_2)$, then all its strongly connected components must consist of a simple cycle. Therefore, according to Lemma 2 and Theorem 1 we have that $G$ is universally stable. □

Using the arguments in the previous proof, Theorem 8 gives the following property.

COROLLARY 1. *A strongly connected digraph $G$ with $n$ vertices is universally stable if and only if $G$ is the directed cycle on $n$ vertices.*

For the case of universal stability of graphs, the basic set of forbidden subgraphs is given in Figure 5(a).

THEOREM 9. *A graph is universally stable if and only if it does not contain as subgraphs any of the graphs in* $\mathcal{E}(H_1) \cup \mathcal{E}(H_2)$.

*Proof.* The "only if" part follows from Theorem 5. If a graph $G$ does not contain as a subgraph any of the graphs in $\mathcal{E}(H_1) \cup \mathcal{E}(H_2)$, then $G$ has no incident edges, so it is universally stable. □

For the case of universal stability the corresponding graph property is the following corollary.

COROLLARY 2. *A graph $G$ is universally stable if and only if all the vertices in $G$ have degree at most* 1.

For the case of undirected-path universal stability of graphs, the basic set of forbidden subgraphs is given in Figure 5(b).

THEOREM 10. *A graph $G$ is undirected-path universally stable if and only if $G$ does not contain as a subgraph any of the graphs in* $\mathcal{E}(F_1) \cup \mathcal{E}(F_2) \cup \mathcal{E}(F_3)$.

*Proof.* The "only if" part follows from Theorem 6. Assume now that $G$ does not contain as a subgraph any of the graphs in $\mathcal{E}(F_1) \cup \mathcal{E}(F_2) \cup \mathcal{E}(F_3)$. In this case $G$ does not contain any edge with multiplicity 3, and all the connected components of $G$ contain at most one cycle. Therefore, all the connected components of $G$ are undirected-path universally stable by Lemma 3. □

For the case of undirected-path universal stability of graphs, the corresponding property is the following corollary.

COROLLARY 3. *A connected graph $G$ is undirected-path universally stable if and only $G$ is a subgraph of a unicyclic graph.*

For the case of simple-path universal stability of graphs, the basic set of forbidden subgraphs is given in Figure 5(c).

THEOREM 11. *A graph $G$ is simple-path universally stable if and only if $G$ does not contain as a subgraph any of the graphs in* $\mathcal{E}(K_1) \cup \mathcal{E}(K_2) \cup \mathcal{E}(K_3)$.

*Proof.* The "only if" part follows from Theorem 7. When $G$ does not contain as subgraphs any of the graphs in $\mathcal{E}(K_1) \cup \mathcal{E}(K_2) \cup \mathcal{E}(K_3)$, all the connected components of $G$ can have at most one cycle with more than two vertices, so they must be subgraphs of a decorated cycle graph. Therefore, using Lemma 6, $G$ is simple-path universally stable. □

COROLLARY 4. *A connected graph $G$ is simple-path universally stable if and only if $G$ is a subgraph of a decorated cycle graph.*

The case of simple-path universal stability is needed to complete the characterization for digraphs. In this case the basic set of forbidden subgraphs is given in Figure 4(b).

THEOREM 12. *A digraph $G$ is simple-path universally stable if and only if $G$ does not contain as a subgraph any of the graphs in $\mathcal{E}(S_1) \cup \mathcal{E}(S_2) \cup \mathcal{E}(S_3) \cup \mathcal{E}(S_4)$.*

*Proof.* As before, the "only if" part follows from Theorem 4. When $G$ excludes the family of forbidden subgraphs, each strongly connected component of $G$ must contain at most one directed cycle with more than two vertices. Therefore all the strongly connected components are subgraphs of a decorated directed cycle graph. Therefore, by Lemma 7, all the strongly connected components are simple-path universally stable, and by Theorem 1, $G$ is simple-path universally stable. ∎

COROLLARY 5. *A strongly connected digraph $G$ is simple-path universally stable if and only if $G$ is a subgraph of a decorated directed cycle graph.*

We have shown that the graph $S_1$ in Figure 4(b) is not simple-path stable under the NTG-LIS protocol. The only graph transformation proposed by Goel in [14] consists of replacing arcs by disjoint directed paths (see Corollaries 2.5 and 2.7 in [14]). Hence, our graph $S_1$ is a minor of the first minor proposed there. We also show that the digraph $S_4$ is not simple-path stable; however, $S_4$ does not contain as minor any of the forbidden minors in the characterization in [14], and therefore must be universally stable according to Goel's result. However, we have shown this is not the case. These facts disprove the characterization proposed by Goel in [14] and they establish the set $\{S_1, S_2, S_3, S_4\}$ (see Figure 4(b)) as the set of forbidden subgraphs characterizing the property of simple-path stability for digraphs.

## 6. Deciding universal stability.

In this section we show that the five cases of universal stability presented in this work can be decided in polynomial time. For undirected graphs we could use the polynomial time algorithm for checking subgraph homeomorphism of the corresponding forbidden subgraphs given in [17]; however, our algorithms are much simpler. Notice that for directed graphs checking subdigraph homeomorphism to a fixed digraph is NP-complete. In particular this is the case for the digraphs $S_1$, $S_2$, and $S_3$ (see [11]). However, the combination of several digraphs and the properties outlined in Corollaries 1, 2, 3, 4, and 5 are, as we will see, easier to test.

THEOREM 13. *The universal stability of a given graph or digraph can be decided in polynomial time.*

*Proof.* According to Corollary 2, to decide universal stability for graphs we only have to check if the graph has two incident edges. To check universal stability for digraphs, following Corollary 1, we have to compute the strongly connected components of the graph and then check whether all of them are just one directed cycle. Both tests can be performed in polynomial time. ∎

For the remaining adversarial models we first need to characterize the graphs that are subgraphs of a unicyclic graph.

LEMMA 8. *A connected graph $G$ with $m$ edges and without multiple edges is a subgraph of a unicyclic graph if and only if any spanning tree of $G$ has $m$ or $m-1$ edges.*

According to the previous lemma, Algorithm 1 checks whether a connected graph $G$, without multiple edges, is a subgraph of a unicyclic graph, in polynomial time. The algorithms to decide universal stability combine this checking with some additional testing for the multiedges.

THEOREM 14. *The undirected-path universal stability of a given graph can be decided in polynomial time.*

*Proof.* Algorithm 2 checks the undirected-path universal stability of a given connected graph $G$ according to Corollary 3. Its total execution time is polynomial. Then,

**Algorithm 1 :** Subgraph of a unicyclic graph

```
INPUT: A connected graph $G$
Compute a spanning tree of $G$
if there are more than one edge left then
  return no
else
  return yes
end if
```

**Algorithm 2 :** Graph undirected-path stability

```
INPUT: A connected graph $G$
if some edge has multiplicity 3 then
  return no
else
  compute the connected components of $G$
  if a connected component has two edges with multiplicity 2 then
    return no
  end if
  || Now all the connected components have at most one edge with multiplicity 2
  if there is a connected component of $G$ that is not a subgraph of a unicyclic graph then
    return no
  else
    return yes
  end if
end if
```

by combining this algorithm with the computation of the connected components of the given graph, we obtain a polynomial time algorithm. ∎

THEOREM 15. *The simple-path universal stability of a given graph can be decided in polynomial time.*

*Proof.* Algorithm 3 checks the simple-path universal stability of a given connected graph $G$ according to Corollary 4. The total execution time is polynomial. Thus, combining this algorithm with the computation of the connected components of the given graph, we can check the property in polynomial time. ∎

THEOREM 16. *The simple-path universal stability of a given digraph can be decided in polynomial time.*

*Proof.* Algorithm 4 checks the simple-path universal stability of a given strongly connected digraph $G$ according to Corollary 5. The total execution time is polynomial. By combining this algorithm with the computation of the strongly connected components of the given digraph, we obtain a polynomial time algorithm. ∎

All the algorithms presented in this section run in polynomial time. Note that the most expensive operations are the computation of the strongly connected components of a digraph, the computation of the connected components of a graph, and the computation of a spanning tree of a connected graph.

**7. Stability under NTG-LIS.** An interesting question concerning stability is that of deciding the stability of a concrete network under a fixed protocol. Since all the instability results in this work hold for the NTG-LIS protocol, we can conclude that

**Algorithm 3 :** Graph simple-path stability

```
INPUT: A connected graph $G$
Let $G'$ be the graph obtained from $G$ by setting all edge multiplicities to one
Compute the connected components of $G'$
**if** there is a connected component $H$ of $G'$ s.t. $H$ is not a subgraph of a unicyclic graph **or** $H$ is a unicyclic graph having $k \geq 3$ vertices in the cycle and with some cycle edge having multiplicity bigger than 1 in $G$ **then**
  return no
**else**
  return yes
**end if**
```

**Algorithm 4 :** Digraph simple-path stability

```
INPUT: A strongly connected digraph $G$
**if** $G$ does not have a directed $k$-cycle with $k \geq 3$ **then**
  return yes
**else**
  compute a directed cycle $C = (v_k, e_1, v_1, \dots , v_{k-1} e_k, v_k)$ in $G$ ($k \geq 3$)
  **if** any of the cycle arcs has multiplicity bigger than one **then**
    return no
  **end if**
  Let $e'_i$ be the arc opposite to $e_i$, $1 \leq i \leq k$
  **if** all the arcs $e'_i$ are present in $G$ and some of them have multiplicity bigger than one **then**
    return no
  **end if**
  Let $G'$ be the digraph obtained by setting the multiplicity of all arcs in $G$ to one and by removing the arcs in $C$ and all the opposite arcs $e'_i$ (if any).
  **if** there are 2-cycle vertices connected by a directed path in $G'$ **then**
    return no
  **else**
    Compute the strongly connected components of $G'$
    **if** a strongly connected component of $G'$ contains a directed $k$-cycle with $k \geq 3$ **then**
      return no
    **else**
      return yes
    **end if**
  **end if**
**end if**
```

testing universal stability is equivalent to testing stability under NTG-LIS. Moreover, deciding whether a network is stable under the NTG-LIS protocol can also be solved in polynomial time. The equivalences hold for all the cases of universal stability considered in this work, as we state in the following theorem.

THEOREM 17. *The following equivalences hold:*

- *A digraph G is stable under* NTG-LIS *if and only if G is universally stable.*
- *A digraph G is simple-path stable under* NTG-LIS *if and only if G is simple-*

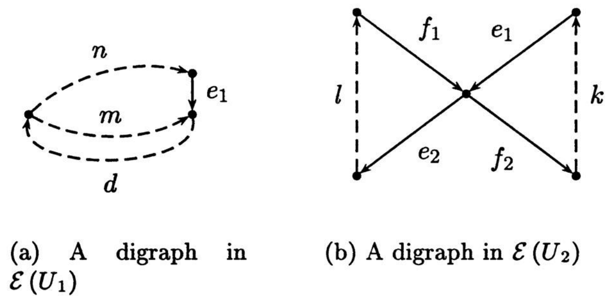

n
m
d
$e_1$
(a) A digraph in $\mathcal{E}(U_1)$
$f_1$
$e_1$
$l$
$k$
$e_2$
$f_2$
(b) A digraph in $\mathcal{E}(U_2)$

FIG. 6. *Family of digraphs formed by extensions of $U_1$ and $U_2$.*

*path universally stable.*

- *A graph G is stable under* NTG-LIS *if and only if G is universally stable.*
- *A graph G is undirected-path stable under* NTG-LIS *if and only if the graph G is undirected-path universally stable.*
- *A graph G is simple-path stable under* NTG-LIS *if and only if G is simple-path universally stable.*

A similar result was obtained in [1] for the universal stability of digraphs under the FFS protocol, for which the same characterization (i.e., in terms of the family of graphs generated from $U_1$ and $U_2$; see Figure 4(a)) is obtained. The technique can be easily extended to show the equivalence of stability under FFS and universal stability in all the models of adversary considered in this work. To the best of our knowledge the complexity of deciding stability under other nonuniversally stable protocols is still open, in particular for FIFO and LIFO. Even though much effort has been devoted to the study of the FIFO protocol, it is still not known whether deciding stability under FIFO is polynomial time decidable. The bottleneck, with regard to the characterization of universal stability, is whether the pair $(U_1, \text{FIFO})$ is stable, as the pair $(U_2, \text{FIFO})$ was shown to be not stable in [2].

## Appendix A. Proof of Theorem 3.

All the instability proofs are based on induction. A set of rounds compose a step of the induction reasoning. The goal is to demonstrate that the number of packets in the system can increase from one step to the next (and, by applying the inductive hypothesis, they can increase indefinitely). The configuration of the system at the end of every step must be the same as the configuration at the beginning of each inductive step (in terms of the type of packets and their location), but with an increased number of packets. In these appendices we reproduce only the inductive step.

LEMMA 9. *The pair $(U_2, \text{NTG-LIS})$ is not stable.*

*Proof.* At the beginning there are $s$ packets that must traverse edge $f_2$. Half of them are of the form $(e_1\ f_2)$, and half are of the form $(e_2\ f_1\ f_2)$. Then the adversary will play injections in four rounds.

*Round* 1. For $s$ steps, the adversary injects $rs$ packets of the form $(f_2\ e_1)$. These injections get mixed with the initial packets at edge $f_2$ and are blocked there because the queuing protocol is NTG.

*Round* 2. For the next $rs$ steps, the adversary injects a set of $r^2s$ packets of the form $(f_2\ e_1\ e_2)$ that are blocked by the remaining packets at $f_2$ from the previous round because the protocol is NTG.

*Round* 3. For the next $r^2s$ steps, the adversary injects $r^3s$ packets of the form

$(e_2)$ and $r^3s$ of the form $(e_1\ f_2)$. A total number of $r^3s$ packets of the from $(e_2)$ will remain at $e_2$. Injections of the form $(e_1\ f_2)$ will meet packets from the previous round at edge $e_1$ and, because the secondary protocol is LIS, they all will be blocked at this edge.

*Round* 4. For the next $r^3s$ steps, the adversary injects $r^4s$ packets of the form $(e_2\ f_1\ f_2)$ and $r^4s$ packets of the form $(e_1)$. The simple injections at $e_1$ block the packets $(e_1\ f_2)$ from the previous round because their distance to destination is shorter. The $(e_2\ f_1\ f_2)$ injections are blocked by the $(e_2)$ packets from the previous round.

At the end of the fourth round there are $r^4s$ packets of the form $(e_2\ f_1\ f_2)$ and $r^4s$ packets of the form $(e_1\ f_2)$. If cycles are allowed, the adversary described above makes the network $U_2$ nonstable when $2r^4s > s$, i.e., at injection rate $r > 0.84089$. □

LEMMA 10. *Any graph in* $\mathcal{E}(U_1)$ *is not stable for* NTG-LIS.

*Proof.* Let $G$ be the graph in $\mathcal{E}(U_1)$ described in Figure 6(a). This graph is obtained from $\mathcal{S}_1$ by replacing the edges by paths. Let us denote by $p_n$, $p_m$, and $p_d$ the path replacing the edges. Let us assume that $n \geq m-1$. The adversary operates in four rounds. Initially, there are $s$ packets wanting to traverse $p_d$ and they are distributed in the following way: $x$ packets of the form $(p_n\ e_1\ p_d)$ and $y$ packets of the form $(p_m\ p_d)$.

*Round* 1. For $s$ steps, the adversary injects $rs$ packets of the form $(p_d\ p_m)$. These injections are blocked by the initial packets because the queuing protocol is NTG, but $rm$ of them are lost. We consider a big enough $s$ to guarantee that a continuous flow arrives to the first edge of $p_d$ after the arrival of the first packet.

*Round* 2. For the next $rs - rm$ steps, the adversary injects a set of $r^2s - r^2m$ packets of the form $(p_d\ p_n\ e_1)$ and $r^2s - r^2m$ packets of the form $(p_m)$. These injections are blocked by the remaining packets from the previous round because the queuing protocol is NTG (or because of the LIS protocol if $n = m-1$). Note that $rd$ injections of the form $(p_m)$ will be lost.

*Round* 3. For the next $r^2s - r^2m$ steps, the adversary injects $r^3s - r^3m$ packets of the form $(p_m)$ that will all be blocked by packets from the previous round. The adversary also injects $r^3s - r^3m$ packets of the form $(p_n\ e_1\ p_d)$ that will collapse with packets from the previous round. These injections will be blocked there because their distance to destination is longer at that point, but $rd$ of them will be lost. At the end of this round there are $r^3s - r^3m - r^2d$ packets of the form $(p_m)$ and $r^3s - r^3m - rd$ packets of the form $(p_n\ e_1\ p_d)$.

*Round* 4. For the next $r^3s - r^3m - r^2d$ steps, the adversary injects $r^4s - r^4m - r^3d$ packets of the form $(p_n\ e_1)$ and $r^4s - r^4m - r^3d$ packets of the form $(p_m\ p_d)$. The injections on $p_n$ will keep some packets of the previous round blocked, and the injections on $p_m$ will be blocked by the remaining packets from the previous round.

At the end of the fourth round, there are $r^4s - r^4m + r^3d - r^2d$ packets of the form $(p_n\ e_1\ p_d)$ and $r^4s - r^4m - r^3d$ packets of the form $(p_m\ p_d)$. The adversary described above uses packets describing cycles and makes the graph $G$ described above nonstable when $2r^4s - 2r^4m - r^2d > s$. Note that $C = 2r^4m + r^2d > 2m + d$ and, for a big enough $s$, an injection rate $r$ can be found such that $2r^4s - C > s$ holds. □

LEMMA 11. *Any graph in* $\mathcal{E}(U_2)$ *is not stable for* NTG-LIS.

*Proof.* Let $G$ be a graph in $\mathcal{E}(U_2)$ described in Figure 6(b). This graph is formed from $\mathcal{S}_2$ by extending its cycles with paths. We call $p_k$ and $p_l$ the additional edges in the two sides, noting that $k$, $l$, or both may be zero. The adversary operates in four rounds. Initially, there are $\frac{s}{2}$ packets of the form $(e_1\ f_2)$ and $\frac{s}{2}$ packets of the form $(e_2\ p_l\ f_1\ f_2)$.

*Round* 1. For $s$ steps, the adversary injects $rs$ packets of the form $(f_2\ p_k\ e_1)$. These injections are blocked by the initial packets because the queuing protocol is NTG. Note that $rl$ of such new packets are lost.

*Round* 2. For the next $rs - rl$ steps, the adversary injects a set of $r^2s - r^2l$ packets of the form $(f_2\ p_k\ e_1\ e_2)$. These injections are blocked by the remaining packets from the previous round because the queuing protocol is NTG.

*Round* 3. For the next $r^2s - r^2l$ steps, the adversary injects $r^3s - r^3l$ packets of the form $(e_2)$ and $r^3s - r^3l$ of the form $(e_1\ f_2)$. These injections will be blocked by packets from the previous round when collapsing at edges $e_2$ and $e_1$, respectively, but $rk$ of them will be lost.

*Round* 4. For the next $r^3s - r^3l - rk$ steps, the adversary injects $r^4s - r^4l - r^2k$ packets of the form $(e_1)$ that will block packets from the previous round at $e_1$ because they are closer to their destination. The adversary also injects $r^4s - r^4l - r^2k$ packets of the form $(e_2\ p_l\ f_1\ f_2)$ that will be blocked by the $(e_2)$ remaining packets from the previous round.

At the end of the fourth round, there are $r^4s - r^4l - r^2k$ packets of the form $(e_2\ p_l\ f_1\ f_2)$ and $r^4s - r^4l - r^2k$ packets of the form $(e_1\ f_2)$. The adversary described above uses packets describing cycles and makes the graph $G$ described above nonstable when $2(r^4s - r^4l - r^2k) > s$. Note that $C = 2(l + k) > 2(r^4l + r^2k)$, and, for a big enough $s$, an injection rate $r$ can be found such that $2r^4s - C > s$ holds. □

## Appendix B. Proof of Theorem 4.

LEMMA 12. *The pair $(S_1,$ NTG-LIS$)$ is not simple-path stable.*

*Proof.* The adversary operates in five rounds. Initially, there are $\frac{s}{2}$ packets of the form $(e_1\ f_1)$ and $\frac{s}{2}$ of the form $(e_2\ f_1)$.

*Round* 1. For $s$ steps, the adversary injects $rs$ packets of the form $(f_1\ f_2)$. These injections get mixed with the initial packets at edge $f_1$ and are blocked there because the protocol is NTG.

*Round* 2. For the next $rs$ steps, the adversary injects a set of $r^2s$ packets of the form $(f_2\ e_2)$ that are blocked at $f_2$ by the packets from the first round because, at that point, they are at longer distance to their destination.

*Round* 3. For the next $r^2s$ steps, the adversary injects $r^3s$ packets of the form $(e_2)$ and $r^3s$ of the form $(f_2\ e_1)$. All these injections are blocked by the remaining packets from the previous round because the secondary protocol is LIS.

*Round* 4. For the next $r^3s$ steps, the adversary injects $r^4s$ packets of the form $(e_1)$ and $r^4s$ packets of the form $(e_2\ f_1)$. A total number of $r^4s$ simple packets will remain at $e_1$. The injections of the form $(e_2\ f_1)$ are blocked at $e_2$ by packets from the previous round because distance to their destination is longer.

*Round* 5. For the next $r^4s$ steps, the adversary injects $r^5s$ packets of the form $(e_1\ f_1)$ and $r^5s$ packets of the form $(e_2)$. The simple packets queued at $e_1$ from the previous round will block the injections introduced at this edge. The simple injections at $e_2$ will block the packets of the form $(e_2\ f_1)$ from the previous round.

At the end of the fifth round, there are $r^5s$ packets of the form $(e_2\ f_1)$ and $r^5s$ packets of the form $(e_1\ f_1)$. The adversary described above uses only packets describing simple paths and makes network $U_1$ nonstable when $2r^5s > s$, i.e., at injection rate $r > 0.87055$. □

LEMMA 13. *The pair $(S_2,$ NTG-LIS$)$ is not simple-path stable.*

*Proof.* The adversary operates in four rounds in this case. Initially, there are $\frac{s}{2}$ packets of the form $(e_{12}\ f)$ and $\frac{s}{2}$ of the form $(e_{22}\ f)$.

*Round* 1. For $s$ steps, the adversary injects $rs$ packets of the form $(f\ e_{21})$. These injections get mixed with the initial packets and are blocked in edge $f$ because the protocol is NTG.

*Round* 2. For the next $rs$ steps, the adversary injects a set of $r^2s$ packets of the form $(f\ e_{11})$ and $r^2s$ packets of the form $(e_{21}\ e_{22})$. The injections made on $f$ are blocked by old packets because in this situation the secondary protocol LIS is applied. When the $(f\ e_{21})$ packets from the previous round reach edge $e_{21}$, they block the injections made on $e_{21}$. So at this round all the injections introduced are blocked.

*Round* 3. For the next $r^2s$ steps, the adversary injects $r^3s$ packets of the form $(e_{22}\ f)$ and $r^3s$ of the form $(e_{11}\ e_{12})$. The injections on $e_{22}$ are blocked by packets of the form $(e_{21}\ e_{22})$ from the previous round. The injections introduced on $e_{11}$ are blocked by packets from the previous round at $e_{11}$ because their destination is farther.

*Round* 4. For the next $r^3s$ steps, the adversary injects $r^4s$ packets of the form $(e_{12}\ f)$ and $r^4s$ packets of the form $(e_{22})$. Injections on $e_{12}$ are blocked by old packets, and injections on $e_{22}$ block old packets of the form $(e_{22}\ f)$.

At the end of the fourth round, there are $2r^4s$ packets queued at $e_{12}$ and $e_{22}$ that want to traverse edge $f$; $r^4s$ are of the form $(e_{12}\ f)$ and $r^4s$ are of the form $(e_{22}\ f)$. The adversary described above uses only simple paths and makes network $S_2$ nonstable when $2r^4s > s$, i.e., at injection rate $r > 0.84089$. $\square$

LEMMA 14. *The pair* $(S_3, \text{NTG-LIS})$ *is not simple-path stable.*

*Proof.* The adversary operates in four rounds. Initially, there are $\frac{s}{2}$ packets of the form $(e_1\ f_{21})$ and $\frac{s}{2}$ of the form $(e_{22}\ f_1\ f_{21})$.

*Round* 1. For $s$ steps, the adversary injects $rs$ packets of the form $(f_{21}\ f_{22})$. These injections get mixed with the initial packets and get blocked in edge $f_{21}$ because the protocol is NTG.

*Round* 2. For the next $rs$ steps, the adversary injects a set of $r^2s$ packets of the form $(f_{22}\ e_1\ e_{21})$ that are blocked at $f_{22}$ by the packets from the first round because, at that point, they are at longer distance to their destination.

*Round* 3. For the next $r^2s$ steps, the adversary injects $r^3s$ packets of the form $(e_{21}\ e_{22})$ and $r^3s$ of the form $(e_1\ f_{21})$. The injections introduced at $e_1$ reach the packets from the previous round at this edge, but they are blocked there because the secondary protocol is LIS. Old packets block the injections introduced at their last edge $e_{21}$.

*Round* 4. For the next $r^3s$ steps, the adversary injects $r^4s$ packets of the form $(e_{22}\ f_1\ f_{21})$ and $r^4s$ packets of the form $(e_1)$. The simple injections on $e_1$ will block the packets queued at this edge from the previous round. Injections of the form $(e_{22}\ f_1\ f_{21})$ are blocked at $e_{22}$ by packets from the previous round because distance to their destination is longer.

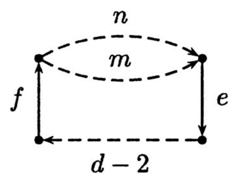
At the end of the fourth round, there are $r^4s$ packets of the form $(e_{22}\ f_1\ f_{21})$ queued at $e_{22}$, and $r^4s$ of the form $(e_1\ f_{21})$ queued at $e_1$. The adversary described above uses only simple paths and makes network $S_3$ nonstable when $2r^4s > s$, i.e., at rate $r > 0.84089$. $\square$

LEMMA 15. *The pair* $(S_4, \text{NTG-LIS})$ *is not simple-path stable.*

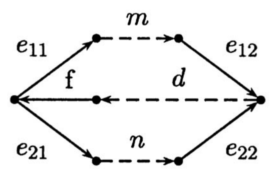
*Proof.* The adversary operates in four rounds. Initially, there are $\frac{s}{2}$ packets of the form $(e_1\ f_{21})$ and $\frac{s}{2}$ of the form $(e_{22}\ f_1\ g_2\ f_{21})$.

*Round* 1. For $s$ steps, the adversary injects $rs$ packets of the form $(f_{21}\ f_{22})$. These injections get mixed with the initial packets and are blocked in edge $f_{21}$ because the protocol is NTG.

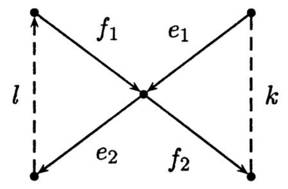
*Round* 2. For the next $rs$ steps, the adversary injects a set of $r^2s$ packets of the



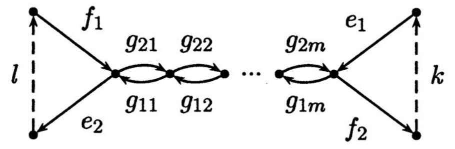
**(a) A digraph in $\mathcal{E}(S_1)$**



**(b) A digraph in $\mathcal{E}(S_2)$**



**(c) A graph in $\mathcal{E}(S_3)$**



**(d) A digraph in $\mathcal{E}(S_4)$**

FIG. 7. *Family of digraphs formed by extensions of $S_1$, $S_2$, $S_3$, and $S_4$.*

form $(f_{22}\ e_1\ g_1)$ that are blocked at $f_{22}$ by the packets from the first round because, at that point, they are at longer distance to their destination.

*Round* 3. For the next $r^2s$ steps, the adversary injects $r^3s$ packets of the form $(g_1\ e_{21}\ e_{22})$ and $r^3s$ of the form $(e_1\ f_{21})$. The injections introduced at $e_1$ reach the packets from the previous round at this edge, but they are blocked there because the secondary protocol is LIS. When the old packets arrive at their last edge $g_1$, they block the injections introduced at that edge. So, at this round, all the injections are blocked.

*Round* 4. For the next $r^3s$ steps, the adversary injects $r^4s$ packets of the form $(e_{22}\ f_1\ g_2\ f_{21})$ and $r^4s$ packets of the form $(e_1)$. The simple injections on $e_1$ will block the packets queued at this edge from the previous round. Injections of the form $(e_{22}\ f_1\ g_2\ f_{21})$ are blocked at $e_{22}$ by packets from the previous round because distance to their destination is longer.

At the end of the fourth round, there are $r^4s$ packets of the form $(e_{22}\ f_1\ g_2\ f_{21})$ queued at $e_{22}$, and $r^4s$ packets of the form $(e_1\ f_{21})$ queued at $e_1$. The adversary described above uses only packets defining simple paths and makes the $S_4$ network nonstable when $2r^4s > s$, i.e., at injection rate $r > 0.84089$. ☐

LEMMA 16. *Any graph in $\mathcal{E}(S_1)$ is not stable for* NTG-LIS.

*Proof.* Let $G$ be the graph in $\mathcal{E}(S_1)$ described in Figure 7(a). There are two vertices $a$ and $b$ with two paths $p_n$ and $p_m$ from $a$ to $b$ and a path $p_d$ from $b$ to $a$, with $d \geq 2$. We assume that $n \geq m$. We call $e$ and $f$ the first and last edges in $p_d$, respectively. By including a whole path (e.g., $p_d$) in the description of the form of a packet, we mean that the path to be followed by the packet must include all of this path. The adversary operates in five rounds. Initially, there are $\frac{s}{2}$ packets of the form $(p_n\ e)$ and $\frac{s}{2}$ of the form $(p_m\ e)$.

*Round* 1. For $s$ steps, the adversary injects $rs$ packets of the form $(p_d)$. These injections get mixed with the initial packets at edge $e$. Except for the first $rm$ injections, the rest are blocked there because the queuing protocol is NTG ($m$ is the time in which a first packet arrives to $e$). We consider $s$ big enough to guarantee that a continuous flow arrives to edge $e$ after the arrival of the first packet.

*Round* 2. For the next $rs - rm$ steps, the adversary injects a set of $r^2s - r^2m$ packets of the form $(f\ p_m)$. These packets will be blocked at $f$ by the packets from the first round because they will be at longer distance to their destination, but notice that $rd$ of them will not be blocked ($d$ is the time needed for the first packet in $e$ to reach $f$).

*Round* 3. For the next $r^2s - r^2m - rd$ steps, the adversary injects $r^3s - r^3m - r^2d$ packets of the form $(p_m)$ and $r^3s - r^3m - r^2d$ packets of the form $(f\ p_n)$. The first injections are blocked because secondary protocol is LIS. The later injections are blocked because $n \geq m$ (or also because of the LIS protocol if $n = m$).

*Round* 4. For the next $r^3s - r^3m - r^2d$ steps, the adversary injects $r^4s - r^4m - r^3d$ packets of the form $(p_n)$, which are blocked due to the LIS effect. Also $r^4s - r^4m - r^3d$ packets of the form $(p_m\ e)$ are injected, which are blocked at the first edge of $p_m$ by packets from the previous round because distance to their destination is longer.

*Round* 5. For the next $r^4s - r^4m - r^3d$ steps, the adversary injects $r^5s - r^5m - r^4d$ packets of the form $(p_n\ e)$, all of which will be blocked by the packets of the form $(p_n)$ from the previous round. The adversary also introduces $r^5s - r^5m - r^4d$ injections of the form $(p_m)$, which will block the packets of the form $(p_m\ e)$ queued since the previous round.

At the end of the fifth round, there are $r^5s - r^5m - r^4d$ packets of the form $(p_n\ e)$ and $r^5s - r^5m - r^4d$ packets of the form $(p_m\ e)$. The adversary described above uses only packets describing simple paths and makes the graph $G$ described above nonstable when $2(r^5s - r^5m - r^4d) > s$. Note that $C = 2(m + d) > 2(r^5m + r^4d)$ is a constant quantity and then, for a big enough $s$, an injection rate $r$ can be found such that $2r^5s - C > s$ holds. $\square$

LEMMA 17. *Any graph in $\mathcal{E}(S_2)$ is not simple-path path stable for* NTG-LIS.

*Proof.* Let $G$ be a digraph in $\mathcal{E}(S_2)$ described in Figure 7(b). This graph is formed from $S_2$ by extending its cycles with paths of length $m$, $n$, and $d$, with $m, n, d \geq 0$. We denote by $p_m$, $p_n$, and $p_d$ the additional path in the respective side. The adversary operates in four rounds. Initially, there are $\alpha$ packets of the form $(e_{12}\ p_d\ f)$ and $\beta$ packets of the form $(e_{22}\ p_d\ f)$, where $\alpha + \beta = s$.

*Round* 1. For $s$ steps, the adversary injects $rs$ packets of the form $(p_d\ f\ e_{21})$. These injections are blocked at the first edge of $p_d\ f$ because the queuing protocol is NTG.

*Round* 2. For the next $rs$ steps, the adversary injects a set of $r^2s$ packets of the form $(f\ e_{11}\ p_m)$ and also $r^2s$ packets of the form $(e_{21}\ p_n\ e_{22})$. Both types of injections will collapse with the remaining packets from the previous round but their destination is always farther; thus they are blocked. Note, however, that $rd$ of each type of packets are lost.[2]

*Round* 3. For the next $r^2s - rd$ steps, the adversary injects $r^3s - r^2d$ packets of the form $(e_{11}\ p_m\ e_{12})$, which are blocked by the remaining packets from the previous round because the protocol is NTG. The adversary also injects $r^3s - r^2d$ packets of the form $(p_n\ e_{22}\ p_d\ f)$, which block $r^3s - r^2d$ of the remaining packets from the previous round because of the same reason.

*Round* 4. For the next $r^3s - r^2d$ steps, the adversary injects $r^4s - r^3d$ packets of the form $(e_{12}\ p_d\ f)$ and also $r^4s - r^3d$ packets of the form $(e_{22})$. The injections on $e_{12}$ are blocked by packets from the previous round, but $rm$ of them are lost. The

[2]More exactly, $r(d + 1)$ of the type $(e_{21}\ p_n\ e_{22})$ are lost, but we round to $rd$ to clarify the proof.

simple injections on $e_{22}$ block packets queued at the previous round, but $rn$ of them are lost.

At the end of the fourth round, there are $r^4 s - r^3 d - rm$ packets of the form $(e_{12}\ p_d\ f)$ queued at $e_{12}$ and $r^4 s - r^3 d - rn$ packets of the form $(e_{22}\ p_d\ f)$ queued at $e_{22}$. The adversary described above uses only packets defining simple paths and makes the digraph described in Figure 7(b) nonstable when $2(r^4 s - r^3 d) - rm - rn > s$. Note that $C = 2d + m + n > 2r^3 d + rm + rn$ and, for a big enough $s$, an injection rate $r$ can be found such that $2r^4 - C > s$ holds. □

LEMMA 18. *Any digraph $G$ in $\mathcal{E}(S_3)$ is not simple-path stable for* NTG-LIS.

*Proof.* Let $G$ be a digraph in $\mathcal{E}(S_3)$ described in Figure 7(c). This graph is formed from $S_3$ by extending its cycles with paths of length $l$ and $k$, $l, k \geq 1$. We denote by $p_k$ and $p_l$ the additional path in the respective side. The adversary operates in four rounds. Initially, there are $\frac{s}{2}$ packets of the form $(e_1\ f_2)$ and $\frac{s}{2}$ packets of the form $(p_l\ f_1\ f_2)$.

*Round* 1. For $s$ steps, the adversary injects $rs$ packets of the form $(f_2\ p_k)$. These injections are blocked by the initial packets because the queuing protocol is NTG. Note that $rl$ of such new packets are lost.

*Round* 2. For the next $rs - rl$ steps, the adversary injects a set of $r^2 s - r^2 l$ packets of the form $(p_k\ e_1\ e_2)$. These injections are blocked by the remaining packets from the previous round because the queuing protocol is NTG.

*Round* 3. For the next $r^2 s - r^2 l$ steps, the adversary injects $r^3 s - r^3 l$ packets of the form $(e_2\ p_l)$ and $r^3 s - r^3 l$ of the form $(e_1\ f_2)$. These injections will be blocked by packets from the previous round when collapsing at edges $e_2$ and $e_1$, respectively, but $rk$ of them will be lost.

*Round* 4. For the next $r^3 s - r^3 l - rk$ steps, the adversary injects $r^4 s - r^4 l - r^2 k$ packets of the form $(e_1)$ that will block packets from the previous round at $e_1$ because they are closer to their destination. The adversary also injects $r^4 s - r^4 l - r^2 k$ packets of the form $(p_l\ f_1\ f_2)$ that will be blocked by the remaining $(e_2\ p_l)$ packets from the previous round.

At the end of the fourth round, there are $r^4 s - r^4 l - r^2 k$ packets of the form $(p_l\ f_1\ f_2)$ and $r^4 s - r^4 l - r^2 k$ packets of the form $(e_1\ f_2)$. The adversary described above uses packets following simple paths and makes the graph $G \in \mathcal{E}(S_3)$ nonstable when $2(r^4 s - r^4 l - r^2 k) > s$. Note that $C = 2(l + k) > 2(r^4 l + r^2 k)$ and, for a big enough $s$, an injection rate $r$ can be found such that $2r^4 s - C > s$ holds. □

LEMMA 19. *Any digraph $G$ in $\mathcal{E}(S_4)$ is not simple-path path stable for* NTG-LIS.

*Proof.* Let $G$ be a digraph in $\mathcal{E}(S_4)$. Notice that if one of the arcs in the 2-cycle is subdivided, $G$ contains as a subgraph a graph in $\mathcal{E}(S_3)$ that by Lemma 18 is NTG-LIS nonstable. Therefore we will consider the case described in Figure 7(d). This graph is formed from $S_4$ by extending its 3-cycles with paths of length $l$ and $k$, $l, k \geq 1$, and its 2-cycle by $m$, $m \geq 1$, 2-cycle successive subdivisions. Let us denote by $p_l$ and $p_k$ the paths of length $l$ and $k$, by $p_m^1$ the path $(g_{11}\ g_{12}\ \dots\ g_{1m})$, and by $p_m^2$ the path $(g_{21}\ g_{22}\ \dots\ g_{2m})$. The adversary operates in four rounds. Initially, there are $\frac{s}{2}$ packets of the form $(e_1\ f_2)$ and $\frac{s}{2}$ of the form $(p_l\ f_1\ p_m^2\ f_2)$.

*Round* 1. For $s$ steps, the adversary injects $rs$ packets of the form $(f_2\ p_k)$. These injections get mixed with the initial packets at edge $f_2$ and are blocked there because the queuing protocol is NTG. Note that $r(m + l)$ of such new packets are lost.

*Round* 2. For the next $rs - r(m + l)$ steps, the adversary injects a set of $r^2 s - r^2(m + l)$ packets of the form $(p_k\ e_1\ g_{1m})$ that are blocked at the first edge of $p_k$ by the packets from the first round because, at that point, they are at longer distance to their destination.

*Round* 3. For the next $r^2s - r^2(m+l)$ steps, the adversary injects $r^3s - r^3(m+l)$ packets of the form $(p_m^1\ e_2\ p_l)$ and $r^3s - r^3(m+l)$ of the form $(e_1\ f_2)$. The injections introduced at $e_1$ reach the packets from the previous round at this edge, but they are blocked there because the secondary protocol is LIS. When the old packets arrive at their last edge $g_{1k}$, they block the injections introduced.

*Round* 4. For the next $r^3s - r^3(m+l)$ steps, the adversary injects $r^4s - r^4(m+l)$ packets of the form $(p_l\ f_1\ p_m^2\ f_2)$ and $r^4s - r^4(m+k)$ packets of the form $(e_1)$. The simple injections on $e_1$ will block the packets queued at this edge from the previous round. The other injections are blocked at the first edge of $p_l$ by packets from the previous round because distance to their destination is longer, but note that $rm$ of them are lost.

At the end of the fourth round, there are $r^4s - r^4(m+l)$ packets of the form $(e_1\ f_2)$ queued at $e_1$ and $r^4s - r^4(m+l) - rm$ packets of the form $(p_l\ f_1\ p_m^2\ f_2)$ queued at the first edge of $p_l$. The adversary described above uses only packets following simple paths and makes the digraph $G$ described in Figure 7(d) nonstable when $2(r^4s - r^4(m+l)) - rm > s$. Note that $C = 2(m+l) > 2r^4(m+l) - rm$ and, for a big enough $s$, an injection rate $r$ can be found such that $2r^4 - C > s$ holds. ∎

**Acknowledgments.** We thank two anonymous referees for their useful comments that improved the readability of the paper.

## REFERENCES

[1] C. Àlvarez, M. Blesa, J. Díaz, A. Fernández, and M. Serna, *The complexity of deciding stability under FFS in the adversarial model*, Inform. Process. Lett., 90 (2004), pp. 261–266.

[2] C. Àlvarez, M. Blesa, J. Díaz, A. Fernández, and M. Serna, *Adversarial models for priority-based networks*, in Proceedings of the 28th International Symposium on Mathematical Foundations of Computer Science (MFCS'03), Bratislava, Slovakia, Lecture Notes in Comput. Sci. 2747, B. Rovan and P. Vojtas, eds. Springer-Verlag, Heidelberg, 2003, pp. 142–151.

[3] M. Andrews, *Instability of FIFO in session-oriented networks*, J. Algorithms, 50 (2004), pp. 232–245.

[4] M. Andrews, B. Awerbuch, A. Fernández, J. Kleinberg, T. Leighton, and Z. Liu, *Universal-stability results for greedy contention-resolution protocols*, J. ACM, 48 (2001), pp. 39–69.

[5] R. Bhattacharjee and A. Goel, *Instability of FIFO at arbitrarily low rates in the adversarial queueing model*, in Proceedings of the 44th Annual IEEE Symposium on Foundations of Computer Science (FOCS'03), Cambridge, MA, IEEE Computer Society, Los Alamitos, CA, 2003, pp. 160–167.

[6] A. Borodin, J. Kleinberg, P. Raghavan, M. Sudan, and D. Williamson, *Adversarial queueing theory*, J. ACM, 48 (2001), pp. 13–38.

[7] M. Bramson, *Instability of FIFO queueing networks*, Ann. Appl. Probab., 4 (1994), pp. 414–431.

[8] R. Cruz, *A calculus for network delay. Part* I: *Network elements in isolation*, IEEE Trans. Inform. Theory, 37 (1991), pp. 114–131.

[9] R. Cruz, *A calculus for network delay. Part* II: *Network analysis*, IEEE Trans. Inform. Theory, 37 (1991), pp. 132–141.

[10] J. Díaz, D. Koukopoulos, S. Nikoletseas, M. Serna, P. Spirakis, and D. Thilikos, *Stability and non-stability of the FIFO protocol*, in Proceedings of the 13th annual ACM Symposium on Parallel Algorithms and Architectures (SPAA'01), Crete Island, Greece, 2001, pp. 48–52.

[11] S. Fortune, J. Hopcroft, and J. Wyllie, *The directed subgraph homeomorphism problem*, Theoret. Comput. Sci., 10 (1980), pp. 111–121.

[12] D. Gamarnik, *Stability of adversarial queues via fluid models*, in Proceedings of the 39th Annual IEEE Symposium on Foundations of Computer Science (FOCS'98), Palo Alto, CA, 1998, pp. 60–70.

[13] D. Gamarnik, *Stability of adaptive and nonadaptive packet routing policies in adversarial queueing networks*, SIAM J. Comput., 32 (2003), pp. 371–385.

[14] A. Goel, *Stability of networks and protocols in the adversarial queueing model for packet routing*, Networks, 37 (2001), pp. 219–224.

[15] D. Koukopoulos, M. Mavronicolas, S. Nikoletseas, and P. Spirakis, *On the stability of compositions of universally stable, greedy contention-resolution protocols*, in Proceedings of the 16th International Symposium on Distributed Computing (DISC'02), Toulouse, France, Lecture Notes in Comput. Sci. 2508, D. Malkhi, ed., Springer, Berlin, 2002, pp. 88–102.

[16] Z. Lotker, B. Patt-Shamir, and A. Rosén, *New stability results for adversarial queuing*, SIAM J. Comput., 33 (2004), pp. 286–303.

[17] N. Robertson and P. Seymour, *Graph minors.* XIII. *The disjoint paths problem*, J. Combin. Theory Ser. B, 63 (1995), pp. 65–110.

[18] A. Rosén, *A note on models for non-probabilistic analysis of packet switching networks*, Inform. Process. Lett., 84 (2002), pp. 237–240.

[19] Z.-L. Zhang, Z. Duan, and Y. Hou, *Fundamental trade-offs in aggregate packet scheduling*, in IEEE 9th International Conference on Network Protocols (ICNP'01), Riverside, CA, IEEE Computer Society, Washington, DC, 2001, pp. 129–137.

SIAM J. COMPUT.
Vol. 34, No. 1, pp. 67–88

# PSEUDORANDOM GENERATORS IN PROPOSITIONAL PROOF COMPLEXITY[*]

MICHAEL ALEKHNOVICH[†], ELI BEN-SASSON[‡], ALEXANDER A. RAZBOROV[§], AND AVI WIGDERSON[¶]

**Abstract.** We call a pseudorandom generator $G_n : \{0,1\}^n \to \{0,1\}^m$ *hard* for a propositional proof system $P$ if $P$ cannot efficiently prove the (properly encoded) statement $G_n(x_1, \dots, x_n) \neq b$ for *any* string $b \in \{0,1\}^m$. We consider a variety of "combinatorial" pseudorandom generators inspired by the Nisan–Wigderson generator on the one hand, and by the construction of Tseitin tautologies on the other. We prove that under certain circumstances these generators are hard for such proof systems as resolution, polynomial calculus, and polynomial calculus with resolution (PCR).

**Key words.** generator, propositional proof complexity, resolution, polynomial calculus

**AMS subject classifications.** 03F20, 03D15

**DOI.** 10.1137/S0097539701389944

## 1. Introduction.

The notion of a pseudorandom generator, originally introduced by Yao [Yao82], has become by now one of the most important concepts in theoretical computer science, penetrating virtually all its subareas. In its simplest form it says the following: a mapping $G_n : \{0,1\}^n \to \{0,1\}^m$ is (computationally) secure with respect to (w.r.t.) some circuit class $\mathcal{C}$ if no "small" circuit $C(y_1, \dots, y_m) \in \mathcal{C}$ can distinguish between the two probabilistic distributions $G_n(\boldsymbol{x})$ and $\boldsymbol{y}$ in the sense that $|\mathbf{P}[C(G_n(\boldsymbol{x})) = 1] - \mathbf{P}[C(\boldsymbol{y}) = 1]|$ is small ($\boldsymbol{x}$ is picked at random from $\{0,1\}^n$, and $\boldsymbol{y}$ is picked at random from $\{0,1\}^m$).

Propositional proof complexity is an area of study that has seen rapid development over the last decade. It plays as important a role in the theory of feasible proofs as the role of the complexity of Boolean circuits plays in the theory of efficient computations. Although the original motivations for this study were in many cases different (and originated from proof-theoretical questions about first-order theories), it turns out after all that the complexity of propositional proofs revolves around the following basic question: What can be *proved* (in the ordinary mathematical sense!) by a prover whose *computational* abilities are limited to small circuits from some circuit class $\mathcal{C}$ (see, e.g., [BP98])? Thus, propositional proof complexity is in a sense complementary to (nonuniform) computational complexity; moreover, there exist extremely rich and productive relations between the two areas (see [Raz96, BP98]).

Given the importance of pseudorandom generators for computational complexity, it is natural to wonder which mappings $G_n : \{0,1\}^n \to \{0,1\}^m$ should be considered

[*]Received by the editors May 31, 2001; accepted for publication (in revised form) April 28, 2003; published electronically October 8, 2004.
http://www.siam.org/journals/sicomp/34-1/38994.html

[†]Moscow State University, Moscow, Russia (mike@mccme.ru). The work of this author was supported by INTAS grant 96-753 and by the Russian Basic Research Foundation.

[‡]Institute of Computer Science, Hebrew University, Jerusalem, Israel (elli@cs.huji.ac.il).

[§]Steklov Mathematical Institute, Moscow, Russia (razborov@genesis.mi.ras.ru). The work of this author was supported by INTAS grant 96-753 and by the Russian Basic Research Foundation; part of this work was done while the author was visiting Princeton University and DIMACS.

[¶]Institute for Advanced Study, Princeton, NJ, and Institute of Computer Science, Hebrew University, Jerusalem (avi@math.ias.edu). The work of this author was supported by grant 69/96 of the Israel Science Foundation, founded by the Israel Academy for Sciences and Humanities. Support for this research was also provided by The Alfred P. Sloan Foundation.

hard from the perspective of proof complexity. In this paper we propose the following paradigm: a generator $G_n : \{0,1\}^n \to \{0,1\}^m$ is hard for some propositional proof system $P$ if and only if for *any* string $b \in \{0,1\}^m$ there is no efficient $P$-proof of the (properly encoded) statement $G(x_1, \dots, x_n) \neq b$ ($x_1, \dots, x_n$ are treated as propositional variables). A similar suggestion is independently made in the recent paper by Krajíček [Kra01].

This definition is very natural: it simply says (to the extent allowed by our framework) that $P$ cannot efficiently prove even the most basic thing about the behavior of $G_n$, namely, that it is not an onto mapping. In fact, one a priori reasonable concern might be precisely whether this exceedingly natural requirement is not too strong, namely, whether nontrivial generators (say, with $m \geq n+1$) can exist at all. This concern is best addressed by exhibiting how several known results fit into our framework; these examples also explain some of our motivations for introducing this concept.

*Example* 1 (Tseitin tautologies). Let $G = (V, E)$ be a connected undirected graph. Consider the ($\mathbb{F}_2$-linear) mapping $T_G : \{0,1\}^E \to \{0,1\}^V$ given by $T_G(\vec{x})_v \overset{\text{def}}{=} \oplus_{e \ni v} x_e$, where $\vec{x} \in \{0,1\}^E$ is a $\{0,1\}$-valued function on edges. Then $b \in \{0,1\}^V$ is not in $\mathrm{im}(T_G)$ if and only if $\oplus_{v \in V} b_v = 1$, and if we properly encode this statement in propositional logic, we arrive exactly at the tautologies introduced by Tseitin in his seminal paper [Tse68]. These tautologies turned out to be extremely useful in propositional proof complexity, and the many strong lower bounds proved for them [Tse68, Urq87, BW99, Gri98, BGIP01, Gri01, ABRW02] never depend on the particular choice of $b \in \{0,1\}^V$. This means that all of them can be viewed as showing that the generators $T_G$ are hard for the corresponding proof system, as long as the graph $G$ itself has good expansion properties.

Tseitin generators $T_G : \{0,1\}^E \to \{0,1\}^V$ make little sense from the computational point of view since the size of the seed $|E|$ is larger than the size of the output $|V|$. Our remaining examples are more satisfactory in this respect.

*Example* 2 (natural proofs). Let $G_n : \{0,1\}^{n^k} \to \{0,1\}^{2^n}$ be any pseudorandom *function generator* that stretches $n^k$ random bits to a Boolean function in $n$ variables viewed as a string of length $2^n$ in its truth-table representation. Assume that $G_n$ is hard w.r.t. $2^{O(n)}$-sized circuits. Razborov and Rudich [RR97] proved that there is no "natural" (in the strict sense also defined in that paper) proof of superpolynomial lower bounds for any complexity class $C$ that can efficiently compute $G_n$. Their argument shows in fact that any natural circuit lower bound techniques fail to prove that a given function $f_n$ does not belong to the image of $G_n$. Stating it equivalently, for *any* function $f_n$ there is no natural proof of the fact that $f_n \notin \mathrm{im}(G_n)$. Although in this result we are primarily interested in the case when $f_n$ is the restriction of SAT (or any other **NP**-complete predicate) onto strings of length $n$, the argument, as in Example 1, absolutely does not depend on the particular choice of $f_n$.

One might argue that natural proofs do not correspond to a propositional proof system at all, and that their definition rather explicitly includes the transition "the proof works for a single $f_n \Rightarrow$ it works for many $f_n$," which provides the link to the ordinary (randomized) definition of a pseudorandom generator. The last two examples illustrate that this drawback sometimes can be circumvented.

*Example* 3 (hardness in presence of feasible interpolation). Let $G_n : \{0,1\}^n \to \{0,1\}^m$ be an arbitrary pseudorandom generator that is hard w.r.t. polynomial size (in $m+n$) circuits, and let $n < m/2$. Following Razborov [Raz95b], let us take bitwise XOR of two independent copies of this generator $G'_n : \{0,1\}^{2n} \to \{0,1\}^m$;

$G'_n(x_1, \ldots, x_n, x'_1, \ldots, x'_n) \overset{\text{def}}{=} G_n(x_1, \ldots, x_n) \oplus G_n(x'_1, \ldots, x'_n)$. Then $G'_n$ is hard for any propositional proof system $P$ which has the property of feasible interpolation (for a definition, see, e.g., [Kra97] or [BP98]).

Indeed, assume for the sake of contradiction that $G'_n$ is easy for a proof system that possesses feasible interpolation. This means that in this system there exists a polynomial size proof of $b \notin \mathrm{im}(G'_n)$ for some string $b \in \{0,1\}^m$. Let $\boldsymbol{r}$ be picked uniformly and at random from $\{0,1\}^m$, and consider the propositional formula encoding the statement $\boldsymbol{r} \notin \mathrm{im}(G_n) \vee \boldsymbol{r} \notin \mathrm{im}(G_n \oplus b)$. The fact $b \notin \mathrm{im}(G'_n)$ implies that this is a tautology and thus, by feasible interpolation, there exists a polynomial size circuit $C$ that given $\boldsymbol{r}$ correctly tells us whether $\boldsymbol{r} \notin \mathrm{im}(G_n)$ or $\boldsymbol{r} \notin \mathrm{im}(G_n \oplus b)$. One of these answers occurs with probability at least 1/2; thus, $C$ can be used to break the generator $G$.

The study of such a keystone concept in computational complexity as pseudorandom generators, but in the new framework of proof complexity, should be interesting in its own right. As suggested by the examples above, we also keep one quite pragmatic goal in mind: we believe that pseudorandomness is methodologically the right way to think of lower bounds in the proof-theoretic setting for really strong proof systems. Whenever we have a generator $G_n : \{0,1\}^n \to \{0,1\}^{n+1}$ which is hard for a propositional proof system $P$, we have lower bounds for $P$. Suppose we manage to increase significantly the number of output bits and construct a polynomial time computable function generator $G_n : \{0,1\}^{n^k} \to \{0,1\}^{2^n}$ that is hard for $P$. Then, similarly to [RR97, Raz95b], we can conclude that the tautologies $\neg Circuit_t(f_n)$, expressing the fact that the function $f_n$ cannot be computed by a circuit of size $t$, do not have efficient $P$-proofs.[1] Our final example shows that, modulo a concrete complexity assumption (namely, that randomness helps in interactive proofs), the tautologies $\neg Circuit_t(f_n)$ are hard for *every* proof system.

*Example* 4. Assume that there is *some* proof system $P$, which efficiently proves $\neg Circuit_t(f_n)$ for *some* Boolean function $f_n$. This proof constitutes an **NP**-certificate of hardness of $f_n$. Using the derandomization machinery of the $NW$-generator [NW94, BFNW93, IW97, IKW01], it follows that for, say, $t = 2^{\epsilon n}$ (with arbitrary $\epsilon > 0$), such a certificate implies that $\mathbf{MA} = \mathbf{NP}$ (and in particular $\mathbf{BPP} \subseteq \mathbf{NP}$).[2]

Put differently, let $G_n$ be the mapping that takes (the encoding of) a circuit of size $t$ to the truth-table of the function computed by it [Raz95a, Appendix C]. Then, assuming $\mathbf{MA} \neq \mathbf{NP}$, we conclude that for an appropriate choice of $t$, there are no efficient proofs of $f_n \notin \mathrm{im}(G_n)$ for any sequence of functions $f_n$. In other words, the generator $G_n$ is hard for any propositional proof system whatsoever!

It should be stressed, however, that some of the authors believe the conclusion far more than the assumption. Nevertheless, the connection illuminates another relationship between computational and proof complexity, and the importance of generators in both.

In this paper we begin by looking at a class of generators inspired by the Nisan–

[1]Note that for $f_n$ an **NP**-complete function and $t$ superpolynomial in $n$, the tautologies $\neg Circuit_t(f_n)$ express the statement $\mathbf{NP} \not\subseteq \mathbf{P}/poly$. The general idea of this reduction to the hardness of generators is similar to the reduction in Example 2: the propositional system cannot prove efficiently that any given $f_n$ does not belong to the image of $G$. However, for every particular system the details of implementation are a little bit different, and one has to be extra careful for weak proof systems.

[2]Follows from $\mathbf{BPP} \subseteq \mathbf{MA}$ of Goldreich and Zuckerman [GZ97]. This weaker conclusion $\mathbf{BPP} \subseteq \mathbf{NP}$ of the existence of efficient proofs for $\neg Circuit_t(f_n)$ was also independently observed by Impagliazzo.

Wigderson generator [NW94] on the one hand, and by Example 1 on the other. Let $A$ be an $(m \times n)$ 0-1 matrix, let $g_1(x_1, \dots, x_n), \dots, g_m(x_1, \dots, x_n)$ be Boolean functions such that $g_i$ essentially depends only on the variables $X_i(A) \overset{\text{def}}{=} \{x_j \mid a_{ij} = 1\}$, and let $G_n : \{0,1\}^n \to \{0,1\}^m$ be given by $G_n(x_1, \dots, x_n) \overset{\text{def}}{=} (g_1(x_1, \dots, x_n), \dots, g_m(x_1, \dots, x_n))$. Nisan and Wigderson [NW94] proved that if $A$ satisfies certain combinatorial conditions (namely, if it is a $(k, s)$-design for suitable choice of parameters), and the functions $g_i$ are computationally hard, then $G_n$ is a good pseudorandom generator in the computational sense. In this paper we study which combinatorial properties of the matrix $A$ and which hardness assumptions imposed on $g_i$ guarantee that the resulting generator $G_n$ is hard for such proof systems as resolution or polynomial calculus.

The framework of proof complexity, however, adds also the third specific dimension that determines hardness properties of $G_n$. Namely, in our examples the base functions $g_i$ are at least supposed to be hard for the circuit class underlying the propositional proof system $P$. Thus, $P$ cannot even express the base functions, and we should encode them using certain extension variables. Using these extension variables, our tautologies can be written as 3-CNFs and thus can be expressed in any proof system. The choice of encoding makes an important part of the framework. We propose three different encodings: functional, circuit, and linear encodings, all natural from both computational and proof complexity viewpoints.

Our results are strong lower bounds for each of these encodings (and appropriate choices of base functions and combinatorial properties of the matrix $A$) in such standard proof systems as resolution, polynomial calculus, and PCR (which combines the power of both). Naturally, the results get weaker as the encoding strength increases.

We strongly believe that this set of tautologies can serve as hard examples for much stronger systems, and specifically that the hardness of the base functions in the generators should be a key ingredient in the proof. This factor is evident in our modest results above, and if extended to stronger systems, it may be viewed as a generalization of the feasible interpolation results, reducing in a sense proof complexity to computational complexity.

The paper is organized as follows. In section 2 we give necessary definitions and describe precisely combinatorial properties of the matrix $A$, hardness conditions imposed on the base functions $g_i$, and types of their encodings needed for our purposes.

Section 3 contains our hardness results for resolution width and polynomial calculus degree that hold for the most general functional encoding similar in spirit to the functional calculus from [ABRW02]. These can be considered as far-reaching generalizations of lower bounds for Tseitin tautologies from [BW99, BGIP01]. We also state here size lower bounds directly implied by our results via the known width/size and degree/size relations.

Section 4 contains a stronger lower bound for the weaker linear encoding. In section 5 we consider the question of maximizing the number of output bits $m = m(n)$ in the generators constructed in the previous sections. For that purpose we show that with high probability a random matrix $A$ has very good expansion properties. The paper is concluded in sections 6 and 7 with a brief account of some recent developments and several remaining open questions.

## 2. Preliminaries.

Let $x$ be a Boolean variable, i.e., a variable that ranges over the set $\{0, 1\}$. A *literal* of $x$ is either $x$ (denoted sometimes as $x^1$) or $\bar{x}$ (denoted sometimes as $x^0$). A *clause* is a disjunction of literals.

For any Boolean function $f : \{0,1\}^n \to \{0,1\}$, $Vars(f)$ will denote the set of its essential variables. An *assignment to* $f$ is a mapping $\alpha : Vars(f) \to \{0,1\}$. A *restriction of* $f$ is a mapping $\rho : Vars(f) \to \{0,1,\star\}$. We denote by $|\rho|$ the number of assigned variables, $|\rho| \overset{\text{def}}{=} |\rho^{-1}(\{0,1\})|$.

The *restriction of* $f$ *by* $\rho$, denoted $f|_\rho$, is the Boolean function obtained from $f$ by setting the value of each $x \in \rho^{-1}(\{0,1\})$ to $\rho(x)$ and leaving each $x \in \rho^{-1}(\star)$ as a variable.

We say that an assignment $\alpha$ *satisfies* $f$ if $f(\alpha) = 1$. For Boolean functions $f_1, \ldots, f_k, g$ we say that $f_1, \ldots, f_k$ *semantically imply* $g$ (denoted $f_1, \ldots, f_k \models g$) if every assignment to $V \overset{\text{def}}{=} Vars(f_1) \cup \cdots \cup Vars(f_k) \cup Vars(g)$ satisfying $f_1, \ldots, f_k$ satisfies $g$ as well (i.e., for all $\alpha \in \{0,1\}^V (f_1(\alpha) = \cdots = f_k(\alpha) = 1 \Rightarrow g(\alpha) = 1)$).

For $n$ a nonnegative integer, let $[n] \overset{\text{def}}{=} \{1, 2, \ldots, n\}$.

Let $A$ be an $(m \times n)$ 0-1 matrix,

$$J_i(A) \overset{\text{def}}{=} \{ j \in [n] \mid a_{ij} = 1 \}, \tag{1}$$

and let $X_i(A) \overset{\text{def}}{=} \{ x_j \mid j \in J_i(A) \}$ and $g_1(x_1, \ldots, x_n), \ldots, g_m(x_1, \ldots, x_n)$ be Boolean functions such that $Vars(g_i) \subseteq X_i(A)$. We will be interested in systems of Boolean equations,

$$\begin{cases} g_1(x_1, \ldots, x_n) = 1, \\ \qquad \cdots \\ g_m(x_1, \ldots, x_n) = 1. \end{cases} \tag{2}$$

We want to state combinatorial properties of the matrix $A$ and hardness conditions of the base functions $g_i$ such that if we properly encode the system (2) as a CNF $\tau(A, \vec{g})$, then this CNF does not possess efficient refutations in a propositional proof system $P$. This sentence has four ingredients, and the necessary definitions for each of them are provided fairly independently.

**2.1. Combinatorial properties of the matrix *A*.** All hardness results proved in this paper will be based on the following combinatorial property generalizing the "edge-expansion" property for ordinary graphs. It is similar to the expansion defined in [BW99].

DEFINITION 2.1. *For a set of rows* $I \subseteq [m]$ *in the matrix* $A$, *we define its* boundary $\partial_A(I)$ *as the set of all* $j \in [n]$ *(called* boundary elements*) such that* $\{ a_{ij} \mid i \in I \}$ *contains exactly one* 1. *We say that* $A$ *is an* $(r, s, c)$-expander *if* $|J_i(A)| \leq s$ *for all* $i \in [m]$ *and for all* $I \subseteq [m] (|I| \leq r \Rightarrow |\partial_A(I)| \geq c \cdot |I|)$.

Let us relate $(r, s, c)$-expanders to several other combinatorial properties already known from the literature.

*Example* 5. For an ordinary graph $G = (V, E)$, its *edge-expansion coefficient* $c_E(G)$ is defined by

$$c_E(G) \overset{\text{def}}{=} \min_{|U| \leq |V|/2} \frac{e(U, V - U)}{|U|},$$

where $e(U, W)$ is the number of edges between $U$ and $W$ (see, e.g., [Alo98] and the references therein). Let $A_G$ be the incidence matrix of a graph $G$ with $m$ vertices and $n$ edges (i.e., $a_{ve} \overset{\text{def}}{=} 1$ if and only if $v \in e$), and let $d$ be the maximal degree of a vertex in $G$. Then $A_G$ is an $(m/2, d, c)$-expander if and only if $c_E(G) \geq c$.

*Example* 6. Let us turn to the combinatorial property originally used in [N91, NW94]. A matrix $A$ is called $(k,s)$-*design* if $|J_i(A)| = s$ for all $i \in [m]$ and

$$|J_{i_1}(A) \cap J_{i_2}(A)| \le k \tag{3}$$

for all $1 \le i_1 < i_2 \le m$. We have the following.

FACT 1. *Every $(k,s)$-design is also an $(r, s, s-kr)$-expander for any parameter $r$.*

*Proof.* Let $I \subseteq [m]$ and $|I| \le r$. Then, due to the property (3), every $J_i(A)$ with $i \in I$ has at most $k \cdot (r-1)$ elements which are *not* in $\partial_A(I)$. Hence it contains at least $s - k \cdot (r-1)$ elements which *are* in $\partial_A(I)$. □

**2.2. Hardness conditions on the base functions.** As explained in the introduction, we are interested in the methods which, given a mapping $G_n : \{0,1\}^n \to \{0,1\}^m$, allow us to show that the fact $b \notin \mathrm{im}(G_n)$ is hard to prove *for any* $b \in \{0,1\}^m$. This means that we want our lower bounds on the refutation complexity to work uniformly not only for system (2) but also for *all* $2^m$ shifted systems,

$$\begin{cases} g_1(x_1, \ldots, x_n) = b_1, \\ \qquad \cdots \\ g_m(x_1, \ldots, x_n) = b_m, \end{cases}$$

$b \in \{0,1\}^m$. We will enforce this simply by requiring that the conditions placed on the base functions $g_1, \ldots, g_m$ be symmetric; i.e., they are satisfied by some $f$ if and only if they are satisfied by $(\neg f)$.

DEFINITION 2.2. *A Boolean function $f$ is $\ell$-*robust *if every restriction $\rho$ such that $f|_\rho = \mathrm{const}$ satisfies $|\rho| \ge \ell$.*

Clearly, this property is symmetric. The most important example of robust functions are the PARITY functions $x_1 \oplus \cdots \oplus x_n \oplus b$, $b \in \{0,1\}$, which are $n$-robust. Our strongest hardness results for the polynomial calculus work only for this specific function.

In fact, $\ell$-robust functions are already very familiar from the computational complexity literature. [FSS84, Ajt83, Yao85, Hås86] proved *computational* lower bounds for $\ell$-robust functions (when $\ell$ is close to $n = |Vars(f)|$) w.r.t. bounded-depth circuits. $(1-\theta)n$-robust functions (where $\theta$ is meant to be a small positive constant) were recently used in [BST98] for obtaining strong lower bounds for branching programs (property "$\mathcal{P}(\theta)$"). In this paper we will use $\ell$-robust functions for constructing generators that are hard for propositional *proof* systems. It is easy to see that most functions on $n$-bits are (say) $0.9n$-robust.

**2.3. Encodings.** Having constructed system (2), we still should decide how to represent it in propositional logic. This step is nontrivial since we are deliberately interested in the case when the propositional system $P$ cannot directly speak of the functions $g_1, \ldots, g_m$. We consider three major possibilities: *functional*, *circuit*, and *linear* encodings; all of them lead to CNFs that in fact without loss of generality (w.l.o.g.) can be further restricted to 3-CNFs (see the proof of Corollary 3.5 below).

**2.3.1. Functional encoding.** This is the strongest possible encoding which is also universal in the sense that it obviously simulates any other conceivable encoding (in fact, it is a "localized" variant of the functional calculus system considered in [ABRW02]).

DEFINITION 2.3. *Let $A$ be an $(m \times n)$* 0-1 *matrix. For every Boolean function $f$ with the property $\exists i \in [m](Vars(f) \subseteq X_i(A))$, we introduce a new* extension *variable $y_f$. Let $Vars(A)$ be the set of all these variables. For the sake of convenience, single variables sometimes will be denoted as $x_i$ instead of $y_{x_i}$. For a clause $C = y_{f_1}^{\epsilon_1} \vee \cdots \vee y_{f_w}^{\epsilon_w}$ in the variables $Vars(A)$, denote by $||C||$ the Boolean function in the variables $x_1, \ldots, x_n$ given by $||C|| \overset{\text{def}}{=} f_1^{\epsilon_1} \vee \cdots \vee f_w^{\epsilon_w}$.*

*Given Boolean functions $\vec{g} = (g_1, \ldots, g_m)$ such that $Vars(g_i) \subseteq X_i(A)$, we denote by $\tau(A, \vec{g})$ the CNF in the variables $Vars(A)$ that consists of all the clauses $C = y_{f_1}^{\epsilon_1} \vee \cdots \vee y_{f_w}^{\epsilon_w}$ for which there exists $i \in [m]$ such that*

$$Vars(f_1) \cup \cdots \cup Vars(f_w) \subseteq X_i(A) \tag{4}$$

*and*

$$g_i \models ||C||. \tag{5}$$

Fact 2. *$\tau(A, \vec{g})$ is satisfiable if and only if system (2) is consistent.*

*Proof.* If $(a_1, \ldots, a_n)$ is a solution to (2), then the assignment which assigns every $y_f$ to $f(a_1, \ldots, a_n)$ is satisfying for $\tau(A, \vec{g})$. For the other direction, let $\vec{b} = (b_f | y_f \in Vars(A))$ be a satisfying assignment for $\tau(A, \vec{g})$. Let $a_j \overset{\text{def}}{=} b_{x_j}$; then, using those axioms $y_{f_1}^{\epsilon_1} \vee \cdots \vee y_{f_w}^{\epsilon_w}$ from $\tau(A, \vec{g})$ for which $f_1^{\epsilon_1} \vee \cdots \vee f_w^{\epsilon_w} \equiv 1$, we can show by induction on the circuit size of $f$ that $b_f = f(a_1, \ldots, a_n)$ for every $y_f \in Vars(A)$. In particular, $g_i(a_1, \ldots, a_n) = b_{g_i} = 1$ (since $\tau(A, \vec{g})$ contains the axiom $y_{g_i}$). Thus, the vector $(a_1, \ldots, a_n)$ is a solution to the system (2). ∎

**2.3.2. Circuit encoding.** This encoding is much more economical in terms of the number of variables than functional encoding. Also, it looks more natural and conforms better to the underlying idea of the extended Frege proof system. The tautologies under this encoding will be polynomial size as long as all $g_i$'s have polynomial size circuits, and thus are potentially hard for Frege (assuming $\mathbf{P}/poly$ contains functions computationally hard for $\mathbf{NC^1}/poly$).

Definition 2.4. *Let $A$ be an $(m \times n)$ 0-1 matrix, and let $C_1, \ldots, C_m$ be single-output Boolean circuits over an arbitrary fixed finite basis, $C_i$ being a circuit in the variables $X_i(A)$. For every $i \in [m]$ and every gate $v$ of the circuit $C_i$, we introduce a special* extension variable *$y_v$, and we identify extension variables corresponding to input gates labeled by the same variable $x_j$. Let $Vars_{\vec{C}}(A)$ be the set of all these extension variables.*

*By $\tau(A, \vec{C})$ we denote the CNF that consists of the following clauses:*

1. *$y_{v_1}^{\bar{\epsilon}_1} \vee \cdots \vee y_{v_d}^{\bar{\epsilon}_d} \vee y_v^{\pi(\epsilon_1, \ldots, \epsilon_d)}$ whenever $v := \pi(v_1, \ldots, v_d)$ is an instruction of one of the circuits $C_1, \ldots, C_m$ and $\epsilon \in \{0,1\}^d$ is an arbitrary vector;*
2. *$y_{v_i}$ when $v_i$ is the output gate of $C_i$ for all $i \in [m]$.*

For a circuit $C$ in the variables $x_1, \ldots, x_n$, let $||C||$ be the Boolean function (in the same variables $x_1, \ldots, x_n$) it computes.

Fact 3. *$\tau(A, \vec{C})$ is satisfiable if and only if the system $||C_1|| = \cdots = ||C_m|| = 1$ is consistent.*

*Proof.* The proof is similar to that of Fact 2. ∎

Fact 4. *There exists a substitution $\sigma$ of variables from $Vars_{\vec{C}}(A)$ by variables from $Vars(A)$ such that $\sigma(\tau(A, \vec{C}))$ is a subset of the set of clauses $\tau(A, ||\vec{C}||)$. In particular, every refutation of $\tau(A, \vec{C})$ in every "reasonable" propositional proof system can be transformed (by applying $\sigma$) into a refutation of $\tau(A, ||\vec{C}||)$ in the same system, which is simpler w.r.t. any "reasonable" complexity measure.*

*Proof.* Let $\sigma(y_v) \overset{\text{def}}{=} y_{||v||}$, where $||v||$ is the function computed by the gate $v$. ∎

**2.3.3. Linear encoding.** This encoding makes sense only when the functions $g_i$ are $\mathbb{F}_2$-linear forms (for historical reasons, this special case of NW-generators is often referred to as *Nisan generators*). In some cases it is more economical than functional encoding in terms of the number of variables. Also, it is much better structured, and we will take advantage of this in section 4.

DEFINITION 2.5. *Let $A$ be an $(m \times n)$ 0-1 matrix. For every $J \subseteq [n]$ such that $\exists i \in [m](J \subseteq J_i(A))$, we introduce a new* extension variable *$y_J$ (with the intended meaning $y_J \sim \oplus_{j \in J} x_j$). Let $Vars_{\oplus}(A)$ be the set of all these variables.*

*Given a Boolean vector $b \in \{0,1\}^m$, we denote by $\tau_{\oplus}(A,b)$ the CNF in the variables $Vars_{\oplus}(A)$ that consists of the following clauses:*

1. *$y_{J_1}^{\epsilon_1} \vee \cdots \vee y_{J_d}^{\epsilon_d}$ whenever there exists $i \in [m]$ such that $J_1 \cup \cdots \cup J_d \subseteq J_i(A)$, the symmetric difference $J_1 \triangle \cdots \triangle J_d$ is empty, and $\bar{\epsilon}_1 \oplus \cdots \oplus \bar{\epsilon}_d = 1$;*
2. *$y_{J_i(A)}^{b_i}$ for all $i \in [m]$.*

Let us denote by $\Sigma_i(A, b_i)$ the Boolean function $\bigoplus_{j \in J_i(A)} x_j \oplus \bar{b}_i$.

FACT 5. *$\tau_{\oplus}(A,b)$ is satisfiable if and only if the system $\Sigma_1(A,b_1) = \Sigma_2(A,b_2) = \cdots = \Sigma_m(A,b_m) = 1$ of linear equations over $\mathbb{F}_2$ is consistent.*

*Proof.* The proof follows from the observation that the conjunction of clauses $y_{J_1}^{\epsilon_1} \vee \cdots \vee y_{J_d}^{\epsilon_d}$ for all $\bar{\epsilon}_1 \oplus \cdots \oplus \bar{\epsilon}_d = 1$ is semantically equivalent to the formula $\bigoplus_{i=1}^{d} y_{J_i} = 0$. □

FACT 6. *There exists a substitution $\sigma$ of variables from $Vars_{\oplus}(A)$ by variables from $Vars(A)$ such that $\sigma(\tau_{\oplus}(A,b))$ is a subset of the set of clauses $\tau(A, \vec{\Sigma}(A,b)) \overset{\text{def}}{=} \tau(A, \Sigma_1(A,b_1), \Sigma_2(A,b_2), \ldots, \Sigma_m(A,b_m))$.*

*Proof.* $\sigma(y_J) \overset{\text{def}}{=} y_{\bigoplus_{j \in J} x_j}$. □

It might be instructive to look at the place occupied in our framework by original Tseitin tautologies (cf. Examples 1 and 5). Let $A_G$ be the incidence matrix of an undirected graph $G$. Then our framework provides three different ways[3] to talk of Tseitin tautologies for graphs $G$ of *arbitrary* degree. All these possibilities are *reasonable* in the sense that although the resulting CNF $\tau$ may have a huge size, it always possesses a sub-CNF of *polynomial* size that is still unsatisfiable. The fourth (unreasonable!) encoding is *primitive*: we allow no extension variables at all and simply represent the functions $\Sigma_i(A, b_i)$ themselves as CNFs of exponential size. For graphs of bounded degree (which is the only case researchers were interested in prior to this paper), the subtle differences between the four encodings disappear, and the whole rich spectrum of various possibilities collapses into ordinary Tseitin tautologies.

In fact, the unreasonable primitive encoding can in principle be considered in the framework of our paper as well. Namely, as we will see in section 5, good $(r,s,c)$-expanders exist even for large constants $s$ (say, $s = 10$). And for constant values of $s$, results proved in any of our reasonable encodings can be translated to the primitive encoding with only constant time increase in the size of the tautology. The primitive encoding, however, is very counterintuitive to the main idea that the base functions $g_i$ should be hard for the circuit class underlying our propositional theory, and to the hope of using these tautologies for stronger proof systems. For this reason we do not discuss in this paper either the primitive encoding itself or the trade-off between the tautology size and the bounds appearing in this encoding when $s \to \infty$.

[3]For circuit encoding we must additionally fix some natural circuits computing the functions $\Sigma_i(A, b_i)$.

## 2.4. Propositional proof systems.

**2.4.1. Resolution.** Resolution is the simplest and probably the most widely studied proof system. It operates with clauses and has one rule of inference called the *resolution rule*:

$$\frac{A \vee x \qquad B \vee \bar{x}}{A \vee B}.$$

A *resolution refutation* of a CNF formula $\tau$ is a resolution proof of the empty clause from the clauses appearing in $\tau$.

The *size* of a resolution proof is the number of different clauses within it. The *width* $w(C)$ *of a clause* $C$ is the number of literals in $C$. The *width* $w(\tau)$ *of a set of clauses* $\tau$ (in particular, the width of a resolution proof) is the maximal width of a clause appearing in this set.

The story of propositional proof complexity began some 35 years ago when, in the seminal paper [Tse68], Tseitin proved superpolynomial lower bounds on the size of any resolution refutation of (what was afterwards called) Tseitin tautologies under one extra regularity assumption on the structure of refutation. Haken [Hak85] was the first to remove this restriction and prove exponential lower bounds for general resolution (for the pigeonhole principle). Urquhart [Urq87] proved exponential lower bounds on the size of general resolution refutations for Tseitin tautologies.

Ben-Sasson and Wigderson [BW99], strengthening a result from [CEI96] (cf. section 2.4.2 below), proved the following width-size relation.

PROPOSITION 2.6. *Let $\tau$ be an unsatisfiable CNF in $n$ variables that has a resolution refutation of size $S$. Then $\tau$ has a resolution refutation of width at most $w(\tau) + O(\sqrt{n \log S})$.*

[BW99] also established a linear lower bound on the width of resolution refutation for Tseitin tautologies. In combination with Proposition 2.6, this gave an alternate (and much simpler) proof of the size lower bound from [Urq87].

**2.4.2. Polynomial calculus and PCR.** Polynomial calculus, introduced by Clegg, Edmonds, and Impagliazzo in [CEI96], is a proof system that models common algebraic reasoning. Despite its algebraic nature, polynomial calculus (PC) turned out extremely useful for studying "pure" propositional proof systems.

PC operates with polynomials $P \in F[x_1, \ldots, x_n]$ for some fixed field $F$; a polynomial $P$ is interpreted as, and often identified with, the polynomial equation $P = 0$. PC has polynomials $x_i^2 - x_i$ ($i \in [n]$) as *default axioms* and has two inference rules:

$$\frac{P_1 \qquad P_2}{\alpha P_1 + \beta P_2};\ \alpha, \beta \in F \quad \text{(scalar addition)}$$

and

$$\frac{P}{x \cdot P} \quad \text{(variable multiplication)}.$$

A *PC refutation* of a set of polynomials $\Gamma$ is a PC proof of 1 from $\Gamma$. The *degree of a PC proof* is the maximal degree of a polynomial appearing within it. The *size of a PC proof* is the total number of monomials within the proof.

First nontrivial lower bounds on the degree of PC refutations were proved by Razborov [Raz98] (for the pigeonhole principle). Grigoriev [Gri98] proved linear lower bounds on the degree of Nullstellensatz refutations (which is a subsystem of PC) for Tseitin tautologies. Finally, Buss et al. [BGIP01] extended the latter bound to arbitrary PC proofs. Following [BGIP01] and the research whose outcome is presented in this paper, Ben-Sasson and Impagliazzo [BI99] further simplified this argument and derived linear degree lower bounds for random CNFs.

[CEI96] proved that small size resolution proofs can be simulated by low degree PC proofs (Proposition 2.6 is a later improvement of this result). [IPS99] observed that the same simulation works also for small size *polynomial calculus* proofs.

Motivated in part by this similarity, [ABRW02] proposed considering the following natural system PCR extending both PC and resolution. PCR operates with polynomials $P \in F[x_1, \dots, x_n, \bar{x}_1, \dots, \bar{x}_n]$, where $\bar{x}_1, \dots, \bar{x}_n$ are treated as new formal variables. PCR has all default axioms and inference rules of PC (including, of course, those that involve new variables $\bar{x}_i$), plus additional default axioms $x_i + \bar{x}_i = 1$ $(i \in [n])$. The *size* and *degree of a PCR proof* are defined in the same way as for PC. It should be noted that there is not much sense in giving a separate definition for the degree of PCR proofs since the linear transformation $\bar{x}_i \mapsto 1 - x_i$ takes a PCR proof to (essentially) a PC proof while preserving degree. This system, however, becomes extremely convenient when it is the number of clauses which matters (see [ABRW02]).

PCR is an extension of PC by definition. Also, PCR extends resolution via the following translation. For a clause $C$, let $C_+$ $[(C_-)]$ be the set of positive (respectively, negative) literals appearing within it. Then a CNF formula $\tau$ gets translated into the set of polynomials $\Gamma_\tau$ defined by $\Gamma_\tau \overset{\text{def}}{=} \{(\prod_{\bar{x} \in C_-} x \cdot \prod_{x \in C_+} \bar{x}) | C \in \tau\}$. Clearly, $\tau$ is satisfiable if and only if $\Gamma_\tau$ has a common root in $F$ satisfying all default axioms

$$x_i^2 = x_i;\ \bar{x}_i^2 = \bar{x}_i;\ x_i + \bar{x}_i = 1. \tag{6}$$

Moreover, it is easy to see that every width $w$ size $S$ resolution refutation of $\tau$ can be transformed into degree $(w+1)$ size $O(nS)$ PCR refutations of the associated set of polynomials $\Gamma_\tau$ (cf. [BG99, section 5]). For ease of notation, we will omit the translation and define a *PCR refutation of a CNF* $\tau$ as a PCR refutation of $\Gamma_\tau$. A *PC refutation* of $\tau$ is a PC refutation of the set of polynomials

$$\Gamma'_\tau \overset{\text{def}}{=} \left\{ \left( \prod_{\bar{x} \in C_-} x \cdot \prod_{x \in C_+} (1 - x) \right) \middle|\ C \in \tau \right\} \tag{7}$$

obtained from $\Gamma_\tau$ by the linear transformation $\bar{x}_i \mapsto 1 - x_i$.

In fact all our lower bounds for PC hold also for PCR, so we will usually use the translation to PCR and prove PCR lower bounds which imply the hardness for PC.

[ABRW02] observed that the two simulations from [CEI96, IPS99] can be merged into one, as follows.

PROPOSITION 2.7. *Let* $\Gamma$ *be a system of polynomials in the variables* $x_1, \dots, x_n$, $\bar{x}_1, \dots, \bar{x}_n$ *that have no common roots in* $F$ *satisfying all default axioms* (6), *and let* $d(\Gamma) \overset{\text{def}}{=} \max\{\deg(P) \mid P \in \Gamma\}$. *Then every size* $S$ *PCR refutation of* $\Gamma$ *can be transformed into another PCR refutation of* $\Gamma$ *that has degree at most* $d(\Gamma) + O(\sqrt{n \log S})$.

**3. Lower bounds on width and degree in the functional encoding.** In this section we establish strong lower bounds on the resolution width and PC degree in the most general functional encoding, and we derive from them some size lower bounds. Our results in this section can be viewed as a far-reaching generalization of the corresponding lower bounds for Tseitin tautologies from [BW99, BGIP01].

But first a word about important and less important parameters. The parameters $s, c, l$ of the defining tautologies will feature in most of the calculations. (Recall that $s$ is the number of 1's in each row of the matrix $A$, which is also the number of arguments to each function $g_i$; $c$ is the expansion factor of the matrix $A$; and $\ell$ will lower bound the robustness of the $g_i$'s.) We will show in section 5 that almost all matrices satisfy $c > 0.9s$. Similarly, most functions satisfy $\ell > 0.9s$. Assuming this, Theorems 3.1 and 3.7 provide $\Omega(r)$ lower bounds on the width of resolution and degree of PC, respectively. (Recall that $r$ is the key parameter defining what size sets expand and can be taken to be essentially $n/s$; see section 5 for details.) Our corollaries for the size lower bounds implied by the width and degree lower bounds will be stated (for simplicity) only for this situation.

Theorem 3.1. *Let $A$ be an $(r, s, c)$-expander of size $(m \times n)$, and let $g_1, \dots, g_m$ be $\ell$-robust functions with $Vars(g_i) \subseteq X_i(A)$, where $c + \ell \geq s + 1$. Then every resolution refutation of $\tau(A, \vec{g})$ must have width $> \frac{r(c+\ell-s)}{2\ell}$.*

*Proof.* The proof follows the ideology developed in [BW99]. We define a measure $\mu$ with subadditive growth on the clauses, and we show that the measure of the empty clause is large ($\mu(0) > r$); hence there must be a clause with medium size measure ($r/2 < \mu(C) \leq r$). We show that such a clause must have large width.

Fix an $(r, s, c)$-expander $A$ of size $(m \times n)$ and $\ell$-robust functions $g_1, \dots, g_m$ with $Vars(g_i) \subseteq X_i(A)$, where $c + \ell \geq s + 1$.

Definition 3.2. *For $C$ a clause in the variables $Vars(A)$, define $\mu(C)$ to be the minimal size of $I \subseteq [m]$ such that the following pair of conditions hold:*

$$\forall y_f^\epsilon \in C \ \exists i \in I \ (Vars(f) \subseteq X_i(A)); \tag{8}$$

$$\{g_i \mid i \in I\} \models ||C||. \tag{9}$$

Claim 3.3.

1. *For a clause $C$ with $r/2 < \mu(C) \leq r$, $w(C) \geq \frac{r(c+\ell-s)}{2\ell}$.*
2. *$\mu(0) > r$.*

*Proof. Part* 1. Let $I$ be a set of minimal size satisfying Definition 3.2. Since $|I| \leq r$, we get $|\partial_A(I)| \geq c \cdot |I|$. Let us partition $I$ into $I_0$, any minimal subset satisfying (8), and $I_1 = I \setminus I_0$. Notice that by the minimality of $I$, removing any row from $I_1$ will ruin property (9).

We claim that for any $i_1 \in I_1$, $J_{i_1}(A)$ has small intersection with $\partial_A(I)$. Namely,

$$|J_{i_1}(A) \cap \partial_A(I)| \leq s - \ell. \tag{10}$$

Indeed, as we note above, $\{g_i \mid i \in I \setminus \{i_1\}\} \not\models ||C||$. Let $\alpha$ be any assignment such that $g_i(\alpha) = 1$ $(i \in I \setminus \{i_1\})$ but $||C||(\alpha) = 0$. Let $\rho$ be the restriction given by

$$\rho(x_j) \overset{\text{def}}{=} \begin{cases} \alpha(x_j) \text{ if } j \notin \partial_A(I) \cap J_{i_1}(A), \\ \star \text{ if } j \in \partial_A(I) \cap J_{i_1}(A). \end{cases}$$

Then, since $\rho$ is totally defined on $Vars(g_i)$ for $i \neq i_1$, and also on $Vars(||C||)$ (by (8) and $i_1 \notin I_0$), we have $g_i|_\rho \equiv 1$ $(i \neq i_1)$ and $C|_\rho \equiv 0$. Hence, using (9), we conclude that $g_{i_1}|_\rho \equiv 0$. Since $g_{i_1}$ is $\ell$-robust and $|J_{i_1}(A)| \leq s$, this implies the desired inequality (10).

Now we may sum up:

$$(11)\qquad \begin{cases} c\cdot|I| \le |\partial_A(I)| \\ \qquad \le s\cdot|I_0| + (s-\ell)|I_1| \\ \qquad = (s-\ell)|I| + \ell\cdot|I_0| \\ \qquad \le (s-\ell)|I| + \ell\cdot w(C), \end{cases}$$

which implies $w(C) \ge \frac{|I|(c+\ell-s)}{\ell}$. Recalling that $|I| > r/2$, we get our bound. Part 1 is proven.

*Part* 2. Suppose the contrary, that is, $\mu(0) \le r$. Then we can repeat the first part of the above argument (since that part did not use the condition $|I| > r/2$) and still get (11). But now $I_0 = \emptyset$, and hence (11) alone implies a contradiction with the expansion property. This proves part 2. ◻

CLAIM 3.4. *Any resolution refutation of $\tau(A, \vec{g})$ must include a clause $C$ with $r/2 < \mu(C) \le r$.*

*Proof.* $\mu$ is subadditive; i.e., if $C$ was derived from $C_0, C_1$ by a single resolution step, then $\mu(C) \le \mu(C_0) + \mu(C_1)$. Additionally, for any axiom $C$, $\mu(C) = 1$. The statement now follows from Claim 3.3(2). ◻

Theorem 3.1 is immediately implied by Claims 3.4 and 3.3(1). ◻

In order to see which *size* lower bounds are implied by Theorem 3.1 via Proposition 2.6, we consider only the typical (and most important) case $c+\ell-s = \Omega(s)$, for which our width lower bound is $\Omega(r)$.

COROLLARY 3.5. *Let $\epsilon > 0$ be an arbitrary fixed constant, let $A$ be an $(r, s, \epsilon s)$-expander of size $(m\times n)$, and let $g_1, \dots, g_m$ be $(1-\epsilon/2)s$-robust functions. Then every resolution refutation of $\tau(A, \vec{g})$ must have size $\exp(\Omega(\frac{r^2}{m\cdot 2^{2^s}}))/2^s$.*

*Proof.* Fix a resolution refutation of $\tau(A, \vec{g})$ that has size $S$. It is easy to see that every axiom in $\tau(A, \vec{g})$ contains a subclause of width $\le 2^s$ which is also an axiom of $\tau(A, \vec{g})$. Moreover, this latter clause can be easily inferred in $O(2^s)$ steps from those axioms in $\tau(A, \vec{g})$ that have width $\le 3$. This allows us to replace the original refutation by a refutation that may have a slightly bigger size $O(S\cdot 2^s)$ but uses only those axioms from $\tau(A, \vec{g})$ that have width $\le 3$. In this new refutation we infer all clauses of $\tau(A, \vec{g})$ that were used in the original refutation from width 3 clauses and then apply the original refutation itself. Hence, by Proposition 2.6, $\tau(A, \vec{g})$ also has a resolution refutation of width $O(\sqrt{|Vars(A)|\cdot\log(S\cdot 2^s)}) \le O(\sqrt{m\cdot 2^{2^s}\cdot\log(S\cdot 2^s)})$. Comparing this with the lower bound of $\Omega(r)$ that comes from Theorem 3.1, we finish the proof of Corollary 3.5. ◻

We can obtain much better size lower bounds (i.e., get rid of the disappointing term $2^{2^s}$ in the denominator) for the circuit encoding. We further confine ourselves to the optimal case when the circuits $C_1, \dots, C_m$ have size $O(s)$.

COROLLARY 3.6. *Let $\epsilon > 0$ be an arbitrary fixed constant, let $A$ be an $(r, s, \epsilon s)$-expander of size $(m\times n)$, and let $C_1, \dots, C_m$ be single-output Boolean circuits over arbitrary fixed finite basis such that $C_i$ is a circuit of size $O(s)$ in the variables $X_i(A)$, and all functions $||C_i||$ are $(1-\epsilon/2)s$-robust. Then every resolution refutation of $\tau(A, \vec{C})$ must have size $\exp(\Omega(\frac{r^2}{ms}))$.*

*Proof.* By Fact 4 and Theorem 3.1, every resolution refutation of $\tau(A, \vec{C})$ must have width $\Omega(r)$. Since $\left|Vars_{\vec{C}}(A)\right| \le O(ms)$, the required bound immediately follows from Proposition 2.6. ◻

Our second major result in this section generalizes the bound from [BGIP01]. Unfortunately, it also inherits all the limitations of their technique: essentially the only base functions $g_1, \ldots, g_m$ we can handle are $\mathbb{F}_2$-linear forms, and for $char(F) = 2$ our approach fails completely (cf. [Gri98]). On the positive side, note that although we do require the linearity of the base functions, the bound itself still holds for the most general *functional* framework.

THEOREM 3.7. *Let $A$ be an $(r, s, c)$-expander of size $(m \times n)$, and let $b_1, \ldots, b_m \in \{0, 1\}$. Then every PCR refutation of $\tau(A, \vec{\Sigma}(A, b))$ over an arbitrary field $F$ with $char(F) \neq 2$ must have degree $\geq \frac{rc}{4s}$.*

*Proof.* As the first step toward proving Theorem 3.7, we show one simple reduction to a lower bound problem about PC refutations in the *original* variables $x_1, \ldots, x_n$. This step is very general and does not depend on the linearity of the base functions $g_i$.

DEFINITION 3.8. *For a Boolean function $f(x_1, \ldots, x_n)$, $P_f(x_1, \ldots, x_n)$ is the (unique) multilinear polynomial such that*

$$P_f(\alpha) = \begin{cases} 0 \text{ if } f(\alpha) = 1, \\ 1 \text{ if } f(\alpha) = 0 \end{cases}$$

*for all $\alpha \in \{0, 1\}^n$.*

LEMMA 3.9. *For any $(m \times n)$ 0-1 matrix $A$ and any functions $g_1, \ldots, g_m$ with $Vars(g_i) \subseteq X_i(A)$, every degree $d$ PCR refutation of $\tau(A, \vec{g})$ can be transformed into a PC refutation of the system*

$$P_{g_1} = \cdots = P_{g_m} = 0 \tag{12}$$

*(in the* original *variables $x_1, \ldots, x_n$) that has degree $\leq s \cdot d$.*

*Proof of Lemma* 3.9. Let us consider some PCR refutation $\pi$ of $\tau(A, \vec{g})$. Substitute in $\pi$ the polynomial $P_{f^\epsilon}(x_1, \ldots, x_n)$ for every variable $y_f^\epsilon$. Since $\deg(P_{f^\epsilon}) \leq s$ for any $f(x_1, \ldots, x_n)$ such that $Vars(f) \subseteq X_i(A)$ for some $i \in [m]$, the degrees of all lines resulting from this substitution are at most $s \cdot d$. Moreover, any axiom from $\tau(A, \vec{g})$, as well as default axioms, gets transformed into a polynomial $P$ such that for some $i \in [m]$ $P$ contains only variables from $X_i(A)$ and is a semantical corollary of $P_{g_i}$ on $\{0, 1\}^{X_i(A)}$. Hence, it can be inferred from $P_{g_i}$ in degree $\leq s$ using only variables from $X_i(A)$. Appending these auxiliary inferences to the beginning of the transformed refutation $\pi$, we obtain the required PC refutation of the system (12). Lemma 3.9 is proved. □

Thus, in order to complete the proof of Theorem 3.7, we should establish the $\frac{rc}{4}$ lower bound on the degree of any PC refutation $\pi$ of the system (12) for $g_i = \Sigma_i(A, b_i)$.

The proof is based on the connection between PC degree and Gaussian width found in [BI99]. With this connection in hand, we may quote here, word for word, Theorem 3.3 from [BI99], plugging in our current parameters.

THEOREM 3.10. *For $A$ an $(r, s, c)$-expander, $\{g_i\}$ linear equations mod 2, and $F$ a field of characteristic $\neq 2$, any PCR refutation of $P_{g_1} = \cdots = P_{g_m} = 0$ has degree $\geq \frac{rc}{4}$.*

Theorem 3.7 follows. □

COROLLARY 3.11. *Let $\epsilon > 0$ be an arbitrary fixed constant, let $A$ be an $(r, s, \epsilon s)$-expander of size $(m \times n)$, and let $b_1, \ldots, b_m \in \{0, 1\}$. Then every PCR refutation of $\tau(A, \vec{\Sigma}(A, b))$ over an arbitrary field $F$ with $char(F) \neq 2$ must have size $\exp(\Omega(\frac{r^2}{m \cdot 2^{2s}}))/2^s$.*

*Proof.* The proof is identical to that of Corollary 3.5, using Proposition 2.7. □

COROLLARY 3.12. *Let $\epsilon > 0$ be an arbitrary fixed constant, let $A$ be an $(r, s, \epsilon s)$-expander of size $(m \times n)$, let $b_1, \ldots, b_m \in \{0, 1\}$, and let $C_1, \ldots, C_m$ be single-output*

*Boolean circuits over an arbitrary fixed finite basis such that $C_i$ is a circuit of size $O(s)$ in the variables $X_i(A)$ that computes the function $\Sigma_i(A, b_i)$. Then every PCR refutation of $\tau(A, \vec{C})$ over an arbitrary field $F$ with $char(F) \neq 2$ must have size* $\exp(\Omega(\frac{r^2}{ms}))$.

*Proof.* The proof is identical to that of Corollary 3.6, using Proposition 2.7. ∎

**4. Size lower bounds for linear encoding.** In this section we show better lower bounds (although our requirement on the expansion rate is somewhat stronger) on the size of PCR refutation for the more structured linear encoding than those provided by Corollaries 3.11 and 3.12. We will apply the random restriction method for killing large clauses rather than directly referring to the general degree/size relation from Proposition 2.7. In this sense our approach is similar in spirit to that of [BP96].

THEOREM 4.1. *Let $A$ be an $(r, s, \frac{3}{4}s)$-expander of size $(m \times n)$, and let $b_1, \dots, b_m \in \{0, 1\}$. Then every PCR refutation of $\tau_\oplus(A, \vec{b})$ over an arbitrary field $F$ with $char(F) \neq 2$ must have size* $\exp(\Omega(\frac{r^2}{m}))$.

*Proof.* As the first step toward proving Theorem 4.1, we show how to get rid of the variables $y_J$ for large (= of size $> s/2$) sets $J$. For technical reasons, we also switch during this step from the linear encoding to the functional one.

DEFINITION 4.2. *For an $(m \times n)$-matrix $A$, the set of variables $Vars_\oplus(A) \subseteq Vars(A)$ consists of those $y_f \in Vars(A)$ for which $f$ has the form $\bigoplus_{j \in J} x_j$. Also let*

$$\widetilde{Vars}_\oplus(A) \stackrel{\text{def}}{=} \left\{ y_{\left(\bigoplus_{j\in J} x_j\right)} \in Vars_\oplus(A) \mid |J| \le s/2 \right\}.$$

*$\tau_\oplus(A, b)$ (respectively, $\tilde{\tau}_\oplus(A, b)$) is the set of those axioms in $\tau(A, \vec{\Sigma}(A, b))$ that contain variables only from $Vars_\oplus(A)$ (respectively, from $\widetilde{Vars}_\oplus(A)$).*

It is worth noting that $\tau_\oplus(A, b)$ possesses the following clean algebraic description: if $g_i = \Sigma_i(a, b_i)$, and $f_1, \dots, f_w$ are $\mathbb{F}_2$-linear forms, then (5) holds if *either* the system of linear equations $f_1 = \bar{\epsilon}_1, \dots, f_w = \bar{\epsilon}_w$ is inconsistent *or* the vector space spanned by these equations contains $\bar{g}_i$.

LEMMA 4.3. *Suppose that $A$ is an $(2, s, \frac{3}{4}s)$-expander. Then every PCR refutation of $\tau_\oplus(A, b)$ can be transformed into a PCR refutation of $\tilde{\tau}_\oplus(A, b)$ that has the same size.*

*Proof of Lemma* 4.3. For every two distinct rows $i_1$ and $i_2$ we have $|\partial_A(\{i_1, i_2\})| \ge \frac{3}{2}s$, which implies $|J_{i_1}(A) \cap J_{i_2}(A)| \le s/2$. Hence, for every $J \subseteq [n]$ with $|J| > s/2$ there can exist at most one row $i \in [m]$ such that $J \subseteq J_i(A)$. Therefore, the mapping

$$y_J \mapsto \begin{cases} y_{\bigoplus\{x_j \mid j \in J\}} & \text{if } |J| \le s/2, \\ y_{\bigoplus\{x_j \mid j \in J_i(A) \setminus J\}} \oplus b_i & \text{if } |J| > s/2 \text{ and } J \subseteq J_i(A) \end{cases}$$

is well-defined. It is easy to see that it takes every axiom from $\tau_\oplus(A, b)$ to an axiom from $\tilde{\tau}_\oplus(A, b)$, which proves Lemma 4.3. ∎

Now, for a monomial $m = y_{f_1}^{\epsilon_1} \dots y_{f_d}^{\epsilon_d}$ in the variables $\widetilde{Vars}_\oplus(A)$, we define its *$A$-degree* $\deg_A(m)$ as the minimal cardinality of a set of rows $I$ with the property $Vars(f_1) \cup \dots \cup Vars(f_d) \subseteq \bigcup_{i \in I} X_i(A)$. The *$A$-degree of a polynomial* is the maximal $A$-degree of a monomial within it, and similarly the *$A$-degree of a PCR* proof is the maximal $A$-degree of a polynomial within it. The following lemma rephrases Theorem 3.7 for $\deg_A$.

LEMMA 4.4. *Let $A$ be an $(r, s, c)$-expander of size $(m \times n)$, and let $b_1, \ldots, b_m \in \{0, 1\}$. Then every PCR refutation of $\tau(A, \vec{\Sigma}(A, b))$ over an arbitrary field $F$ with $char(F) \neq 2$ must have $A$-degree $\geq \frac{rc}{4s}$.*

*Proof of Lemma* 4.4. The only difference from Theorem 3.7 is that we consider here $A$-degree instead of an ordinary one. It is easy to see by inspection that this change does not affect the reduction in Lemma 3.9, and the same proof applies here as well. $\square$

Lemmas 4.3 and 4.4 determine the strategy of the rest of the proof (cf. [BP96]). We want to hit the prospective refutation of $\tilde{\tau}_{\oplus}(A, b)$ by a random restriction $\boldsymbol{\rho}$ in such a way that $\boldsymbol{\rho}$ preserves the structure of $\tau(A, \vec{\Sigma}(A, b))$ and, if the size of the original refutation is small, with a high probability also kills all monomials in the variables $\widetilde{Vars}_{\oplus}(A)$ that have high $A$-degree.

DEFINITION 4.5. *For a set of rows $I$, let us denote by $M_I$ the set of all restrictions $\rho$ such that $\rho^{-1}(\{0, 1\}) = \bigcup_{i \in I} X_i(A)$ and $\rho$ satisfies all equations $\Sigma_i(A, b_i) = 1$ for all $i \in I$.*

Note that if $|I| \leq r$, then, since $A$ is an $(r, s, \frac{3}{4}s)$-expander, the linear forms $\bigoplus \{x_j \mid x_j \in X_i(A)\} = \Sigma_i(A, b_i) \oplus \bar{b}_i$ $(i \in I)$ are linearly independent (because each of its subsets has a form that contains a boundary variable) and thus $M_I$ is a nonempty linear subspace.

Let $A|_I$ be the result of removing from the matrix $A$ all rows $i \in I$ and all columns $j \in \bigcup_{i \in I} J_i(A)$. Any restriction $\rho \in M_I$ can be naturally extended to the variables from $Vars(A)$ by letting $\rho(y_f) \overset{\text{def}}{=} y_{f|_\rho}$. $\rho$ takes variables from $Vars(A)$ to variables from $Vars(A|_I)$. Moreover, those $y_f$ for which $\exists i \in I$ $(Vars(f) \subseteq X_i(A))$ are set to a constant. Finally, $\rho$ always takes axioms from $\tau(A, \vec{g})$ to axioms from $\tau(A|_I, \vec{g}|_\rho)$. The only remaining problem is that $A|_I$ may not inherit good expansion properties: it is easy to get an example showing that it may even contain an empty row! We circumvent this difficulty by further removing all rows that have large intersection with $\bigcup_{i \in I} J_i(A)$ and show in the following lemma that this can always be done in an efficient manner.

LEMMA 4.6. *Let $A$ be an $(r, s, c)$-expander. Then every set of rows $I$ with $|I| \leq r/2$ can be extended to a larger set of rows $\hat{I} \supseteq I$ such that $|\hat{I}| \leq 2 \cdot |I|$ and $A|_{\hat{I}}$ is an $(r, s, 3c - 2s)$-expander.*

*Proof of Lemma* 4.6. Let us recursively add to $I$ new rows (one row $i_0$ at a time) with the property $\left|J_{i_0}(A) \cap \left(\bigcup_{i \in I'} J_i(A)\right)\right| > 2(s - c)$, where $I'$ is the current value of $I$. We claim that this process will terminate (i.e., no new row can be added) in less than $|I|$ steps.

Suppose the contrary, and let $\hat{I}$ be the set of cardinality $2 \cdot |I|$ reached after $|I|$ steps. Then every row $i_0 \in \hat{I} \setminus I$ contains less than $|J_{i_0}(A) - 2(s - c)| \leq (2c - s)$ boundary elements from $\partial_A(\hat{I})$. Hence, $|\partial_A(\hat{I})| < s \cdot |I| + (2c - s) \cdot |I| = 2c \cdot |I|$, a contradiction.

We choose as our $\hat{I}$ the result of termination of this process. Let $I_0$ be a set of rows in $A|_{\hat{I}}$ (i.e., $I_0 \cap \hat{I} = \emptyset$) of cardinality at most $r$. Then $\partial_{A|_{\hat{I}}}(I_0) = \partial_A(I_0) \setminus \bigcup_{i \in \hat{I}} J_i(A)$. Since for every $i \in I_0$, $\left|J_{i_0}(A) \cap \left(\bigcup_{i \in \hat{I}} J_i(A)\right)\right| \leq 2(s - c)$, we have the bound $\left|\partial_{A|_{\hat{I}}}(I_0)\right| \geq |\partial_A(I_0)| - 2(s - c) \cdot |I_0| \geq c \cdot |I_0| - 2(s - c)|I_0| = (3c - 2s) \cdot |I_0|$. Lemma 4.6 is proved. $\square$

Now we are ready to finish the proof of Theorem 4.1. Fix a PCR refutation $\pi$ of $\tilde{\tau}_{\oplus}(A, b)$. Assume w.l.o.g. that 18 divides $r$, and pick at random a set of rows $\boldsymbol{I}$ of cardinality $r/3$ (we are using boldface to stress that it is a random variable).

Choose arbitrarily $\hat{\boldsymbol{I}} \supseteq \boldsymbol{I}$ according to Lemma 4.6, i.e., such that $|\hat{\boldsymbol{I}}| \leq \frac{2r}{3}$ and $A|_{\hat{\boldsymbol{I}}}$ is an $(r, s, s/4)$-expander. Pick $\boldsymbol{\rho} \in M_{\hat{\boldsymbol{I}}}$ at random, and apply this restriction to our PCR refutation $\pi$. This will produce a PCR refutation $\boldsymbol{\rho}(\pi)$ of $\tilde{\tau}_{\oplus}(A|_{\hat{\boldsymbol{I}}}, \boldsymbol{\rho}(\vec{\Sigma}(A, b)))$. By Lemma 4.4 (with $c = s/4$), $\boldsymbol{\rho}(\pi)$ must contain a nonzero monomial $\boldsymbol{\rho}(m)$ of $A|_{\hat{\boldsymbol{I}}}$-degree $> r/18$. Thus, $\pi$ contains a monomial $m$ that has $A$-degree $> r/18$ and *is not killed by* $\boldsymbol{\rho}$. In order to finish the proof, we only have to estimate from above the probability $\mathbf{P}[\boldsymbol{\rho}(m) \neq 0]$ for every individual monomial $m$ with $\deg_A(m) > r/18$.

Fix any such $m = y_{f_1}^{\epsilon_1} \dots y_{f_d}^{\epsilon_d}$, and recall that $f_1, \dots, f_d$ are $\mathbb{F}_2$-linear forms of weight $\leq s/2$. W.l.o.g. assume that $f_1, \dots, f_t$ form a linear basis of the space $Span(f_1, \dots, f_d)$. Then $\bigcup_{\nu=1}^{t} Vars(f_\nu) = \bigcup_{\nu=1}^{d} Vars(f_\nu)$ and, therefore, $\deg_A(y_{f_1}^{\epsilon_1} \dots y_{f_t}^{\epsilon_t}) = \deg_A(m) > r/18$. Hence, w.l.o.g. we can assume from the very beginning that $f_1, \dots, f_d$ are linearly independent.

Let us now introduce one variation of the notion of $A$-degree. Namely, for $m = y_{f_1}^{\epsilon_1} \dots y_{f_d}^{\epsilon_d}$, let $\deg'_A(m)$ be the minimal cardinality of a set of rows $I$ such that these rows "cover" $m$ in the stronger sense for all $\nu \in [d] \exists i \in I (Vars(f_\nu) \subseteq X_i(A))$. Clearly, $\deg_A(m) \leq \deg'_A(m)$ (and $\leq \deg(m)$). Also, $\deg'_A$ is "continuous" in the sense that for every monomial $m$, and for every variable $y_f^\epsilon$, $\deg'_A(m) \leq \deg'_A(m \cdot y_f^\epsilon) \leq \deg'_A(m)+1$. Therefore, we can gradually remove variables from the monomial $m$, one variable at a time, until we find in it a submonomial $m'$ such that $\deg'_A(m')$ is *exactly* equal to $r/18$. For ease of notation, assume w.l.o.g. that $\deg'_A(m) = r/18$ for the original monomial $m$.

Fix now any set of rows $I_0$ with $|I_0| = r/18$ and such that

$$\forall \nu \in [d]\ \exists i \in I_0\ (Vars(f_\nu) \subseteq X_i(A)). \tag{13}$$

We estimate the probability $\mathbf{P}[\boldsymbol{\rho}(m) \neq 0]$ as follows:

$$\mathbf{P}[\boldsymbol{\rho}(m) \neq 0] \leq \mathbf{P}\left[|I_0 \cap \boldsymbol{I}| < \frac{r^2}{100m}\right] + \max_{\substack{|I| = r/3 \\ |I_0 \cap I| \geq \frac{r^2}{100m}}} \mathbf{P}[\,\boldsymbol{\rho}(m) \neq 0\,|\,|\boldsymbol{I} = I|\,].$$

Since $|I_0| = r/18$ and $|\boldsymbol{I}| = r/3$, we can estimate the first term by Chernoff inequality as

$$\mathbf{P}\left[|I_0 \cap \boldsymbol{I}| < \frac{r^2}{100m}\right] \leq \exp\left(-\Omega\left(r^2/m\right)\right). \tag{14}$$

For estimating the second term, fix any individual $I$ such that $|I| = r/3$ and $|I_0 \cap I| \geq \frac{r^2}{100m}$, and let $\hat{I} \supseteq I$ be a corresponding set of rows satisfying the conclusion of Lemma 4.6. We want to estimate $\mathbf{P}\left[\boldsymbol{\rho}_{\hat{I}}(m) \neq 0\right]$, where $\boldsymbol{\rho}_{\hat{I}}$ is picked at random from $M_{\hat{I}}$ (thus, $\boldsymbol{\rho}_{\hat{I}}$ is a random variable that results from $\boldsymbol{\rho}$ after revealing $\hat{\boldsymbol{I}} = \hat{I}$).

Let $I'_0 = I_0 \cap \hat{I}$, $I'_0 = \{i_1, \dots, i_\ell\}$; $\ell \geq r^2/100m$. Since $I_0$ is minimal with the property (13), for every $\nu \in [\ell]$ we can choose $f \in \{f_1, \dots, f_d\}$ such that $Vars(f) \subseteq X_{i_\nu}(A)$ but $Vars(f) \not\subseteq X_i(A)$ for any other $i \in I_0$. Hence, we can assume w.l.o.g. that $Vars(f_\nu) \subseteq X_{i_\nu}(A)$ for $\nu = 1, \dots, \ell$.

Now, let $V_0 \overset{\text{def}}{=} Span(f_1, \dots, f_\ell)$ be the $\mathbb{F}_2$-linear space generated by the linear functions $f_1, \dots, f_\ell$, and let $\hat{V} \overset{\text{def}}{=} Span\left(\{\bigoplus_{j \in J_i(A)} x_j | i \in \hat{I}\}\right)$. $\mathbf{P}\left[\boldsymbol{\rho}_{\hat{I}}(m) \neq 0\right] \leq \mathbf{P}[\boldsymbol{\rho}_{\hat{I}}(y_{f_1}^{\epsilon_1} \dots y_{f_\ell}^{\epsilon_\ell}) \neq 0]$, and the latter probability is less than or equal to $2^{-(V_0 : V_0 \cap \hat{V})}$ (here $(V_0 : V_0 \cap \hat{V}) \overset{\text{def}}{=} \dim(V_0) - \dim(V_0 \cap \hat{V})$ is the standard codimension of linear spaces). To see this, note that $\boldsymbol{\rho}_{\hat{I}}$ gives $\{0,1\}$-values to all variables of $f_1,\ldots,f_\ell$. Let $k = (V_0 : V_0 \cap \hat{V})$. We can choose $k$ linear forms $f_{i_1}, f_{i_2}, \ldots, f_{i_k} \in \{f_1,\ldots,f_\ell\}$ such that the family $f_{i_1},\ldots,f_{i_k}$ is linearly independent modulo $\hat{V}$. Then the values $\boldsymbol{\rho}_{\hat{I}}(f_{i_1}),\ldots,\boldsymbol{\rho}_{\hat{I}}(f_{i_k})$ are independent, and each equal 0 with probability 1/2. Thus, the probability that no $y_{f_i}$ is killed is less than or equal to $2^{-k}$.

Clearly, $2^{-(V_0:V_0\cap\hat{V})} = 2^{\dim(V_0\cap\hat{V})-\ell}$. Hence, we only have to upper bound $\dim(V_0\cap\hat{V})$. Let us denote by $\hat{I}^+$ the set of all rows $i\in\hat{I}$ which appear with coefficient $\alpha_i \neq 0$ in *at least one* sum of the form

$$\bigoplus_{i\in\hat{I}} \alpha_i \cdot \left( \bigoplus_{j\in J_i(A)} x_j \right) \tag{15}$$

that happens to belong to $V_0$, and let $\hat{V}^+ \overset{\text{def}}{=} Span(\{\bigoplus_{j\in J_i(A)} x_j | i \in \hat{I}^+\})$. Then $V_0\cap\hat{V} \subseteq V_0\cap\hat{V}^+$ by our choice of $\hat{I}^+$, and $\dim(V_0\cap\hat{V}) \le \dim(\hat{V}^+) \le |\hat{I}^+|$.

In order to bound from above $|\hat{I}^+|$, we apply the expansion property to $I_0'\cup\hat{I}^+$ (its cardinality does not exceed $r/18+2r/3<r$). We get $|\partial_A(I_0'\cup\hat{I}^+)| \ge \frac{3}{4}s\cdot|I_0'\cup\hat{I}^+|$. Note that rows from $\hat{I}^+\setminus I_0'$ may not contain elements from $\partial_A(I_0'\cup\hat{I}^+)$ at all; otherwise, the corresponding variable would not cancel out in the sum (15), and this would prevent the latter from being in $V_0$ (note that for any form $f\in V_0$, $Vars(f) \subseteq \bigcup_{i\in I_0'} X_i(A)$).

The key observation is that every row $i_\nu$ from $\hat{I}^+\cap I_0'$ may also contain only a relatively small number of boundary elements, namely, at most $(s/2)$. Indeed, $|Vars(f_\nu)| \le s/2$ (see Definition 4.2). Therefore, if $J_{i_\nu}$ would have contained $> s/2$ boundary elements, then at least one boundary variable $x_j \in X_{i_\nu}(A)$ would not belong to $Vars(f_\nu)$, and would once more prevent the sum (15) from lying in $V_0$ (since $j$ belongs to the boundary, $x_j$ may not occur in other forms appearing in this sum).

Summing up the above remarks, we have the upper bound $|\partial_A(I_0'\cup\hat{I}^+)| \le s\cdot|I_0'\setminus\hat{I}^+| + \frac{s}{2}\cdot|I_0'\cap\hat{I}^+|$. Comparing it with the lower bound given by expansion, we get

$$\frac{3}{4}s\cdot|I_0'\cup\hat{I}^+| \le |\partial_A(I_0'\cup\hat{I}^+)| \le s\cdot|I_0'\setminus\hat{I}^+| + \frac{s}{2}\cdot|I_0'\cap\hat{I}^+|,$$

$$\frac{3}{4}s\left(|\hat{I}^+| + |I_0'\setminus\hat{I}^+|\right) \le s\cdot|I_0'\setminus\hat{I}^+| + \frac{s}{2}\cdot|I_0'\cap\hat{I}^+|,$$

and

$$\frac{3}{4}s|\hat{I}^+| \le \frac{1}{4}s\cdot|I_0'\setminus\hat{I}^+| + \frac{s}{2}\cdot|I_0'\cap\hat{I}^+|,$$

which implies $|\hat{I}^+| \le \frac{2}{3}|I_0'| = \frac{2\ell}{3}$.

Therefore, $\dim(V_0\cap\hat{V}) \le \frac{2\ell}{3}$ and $\mathbf{P}\left[\boldsymbol{\rho}_{\hat{I}}(m)\neq 0\right] \le 2^{-\ell/3} \le \exp\left(-\Omega\left(r^2/m\right)\right)$. Together with (14) this implies $\mathbf{P}[\boldsymbol{\rho}(m)\neq 0] \le \exp\left(-\Omega\left(r^2/m\right)\right)$. Hence, $\pi$ must contain at least $\exp\left(\Omega\left(r^2/m\right)\right)$ monomials (of $A$-degree $\ge r/18$) since otherwise we could find a restriction $\rho$ that kills all of them, contrary to Lemma 4.4. The proof of Theorem 4.1 is complete. $\square$

**5. Existence of strong expanders and hard generators.** All our hardness results in the previous two sections are based upon the notion of an $(r, s, c)$-expander. As we noted in the introduction, one of our eventual goals is to be able to stretch $n$ seed bits to as many output bits $m$ as possible so that the resulting generator is hard for as strong propositional proof systems $P$ as possible. In this section we will see what I/O ratio we can achieve with the results from the two previous sections.

All explicit constructions of $(r, s, c)$-expanders that we know of are based upon Examples 5 and 6 from section 2.1. Unfortunately, the resulting expanders turn out to be virtually useless for our purposes since they cannot even break the barrier $m = n$. Let us turn instead to a simple probabilistic argument. We note that in the context of proof complexity, there is not much advantage in having explicit constructions of hard tautologies over existence proofs.

THEOREM 5.1. *For any parameters $s, n$ there exists an $\left(\Omega(n/s) \cdot n^{-O(1/s)}, s, \frac{3}{4}s\right)$-expander of size $(n^2 \times n)$.*

*Proof.* Let us construct a random $(n^2 \times n)$ matrix $\boldsymbol{A}$ as follows. For every $i \in [n^2]$, let $J_i(\boldsymbol{A}) \overset{\text{def}}{=} \{\boldsymbol{j_{i1}}, \ldots, \boldsymbol{j_{is}}\}$, where all $\boldsymbol{j_{i\nu}}$ $(i \in [n^2], \nu \in [s])$ are picked from $[n]$ independently and at random (in fact, we would also obtain the same result by letting $J_i(\boldsymbol{A})$ be uniformly and independently distributed over all $s$-subsets of $[n]$, but with our choice of $J_i(\boldsymbol{A})$ calculations become simpler). We wish to show that

$$\mathbf{P}[\boldsymbol{A} \text{ is not an } (r, s, 3s/4)\text{-expander}] < 1$$

for some $r \geq \Omega(n/s) \cdot n^{-O(1/s)}$. Let $p_\ell$ be the probability that any given $\ell$ rows of the matrix $\boldsymbol{A}$ violate the expansion property. Then, clearly,

$$\mathbf{P}[\boldsymbol{A} \text{ is not an } (r, s, 3s/4)\text{-expander}] \leq \sum_{\ell=1}^{r} n^{2\ell} p_\ell. \tag{16}$$

Fix an arbitrary $I$ of cardinality $\ell \leq r$. Since every column $j \in \bigcup_{i \in I} J_i(\boldsymbol{A}) \setminus \partial_{\boldsymbol{A}}(I)$ belongs to at least two sets $J_i(\boldsymbol{A})$, we have the bound $\left|\bigcup_{i \in I} J_i(\boldsymbol{A})\right| \leq |\partial_{\boldsymbol{A}}(I)| + \frac{1}{2} \cdot \left(\sum_{i \in I} |J_i(\boldsymbol{A})| - |\partial_{\boldsymbol{A}}(I)|\right) \leq \frac{1}{2}(s\ell + |\partial_{\boldsymbol{A}}(I)|)$. Hence $\partial_{\boldsymbol{A}}(I) < \frac{3}{4}s\ell$ implies also $\left|\bigcup_{i \in I} J_i(\boldsymbol{A})\right| < \frac{7}{8}s\ell$, and $p_\ell$ can be estimated by the union bound as

$$p_\ell \leq \binom{n}{\frac{7}{8}s\ell} \cdot \left(\frac{7s\ell}{8n}\right)^{s\ell} \leq (O(s\ell/n))^{s\ell/8} \leq (O(sr/n))^{s\ell/8}.$$

Substituting this bound into (16), we obtain

$$\mathbf{P}[\boldsymbol{A} \text{ is not an } (r, s, 3s/4)\text{-expander}] \leq \sum_{\ell=1}^{r} n^{2\ell} \cdot \left(O\left(\frac{sr}{n}\right)\right)^{s\ell/8}. \tag{17}$$

The sum in the right-hand side is the geometric progression with the base $n^2 \cdot (O(sr/n))^{\Omega(s)}$. Hence, if $r = (\epsilon n/s) \cdot n^{-1/s\epsilon}$ for a sufficiently small $\epsilon > 0$, the right-hand side of (17) is less than $(1/2)$, which completes the proof of Theorem 5.1. $\square$

COROLLARY 5.2. *There exists a family of $(m \times n)$ matrices $A^{(m,n)}$ such that for every $b = (b_1, \ldots, b_m) \in \{0,1\}^m$, any PCR refutation of $\tau(A^{(m,n)}, \vec{\Sigma}(A^{(m,n)}, b))$ over an arbitrary field with $char(F) \neq 2$ must have size $\exp(\frac{n^{2-O(1/\log\log n)}}{m})$.*

*Proof.* Since for $m \geq n^2$ the bound becomes trivial, we can assume that $m \leq n^2$. Apply Theorem 5.1 with $s = \frac{1}{2}\log_2 \log_2 n$, and cross out in the resulting matrix all rows but (arbitrarily chosen) $m$. This will result in an $(r, s, \frac{3}{4}s)$-expander $A^{(m,n)}$ of size $(m \times n)$, where $r \geq n^{1-O(1/\log\log n)}$. Now we only have to apply Corollary 3.11 and notice that $2^{2^s} = 2^{\sqrt{\log n}} \leq n^{1/\log\log n}$. ◻

Corollary 5.2 shows that in the functional encoding we can stretch $n$ random bits to $n^{2-O(1/\log\log n)}$ bits so that this generator will be hard for (polynomial size) PCR proofs over an arbitrary field $F$ with $char(F) \neq 2$. In particular, it is hard for resolution.

COROLLARY 5.3. *There exists a family of $(m \times n)$ matrices $A^{(m,n)}$ such that $\left|J_i\left(A^{(m,n)}\right)\right| \leq \log_2 n$ for all $i \in [m]$, and for every $b = (b_1, \ldots, b_m) \in \{0,1\}^m$ we have the following bounds:*

1. *Let $C_1, \ldots, C_m$ be single-output Boolean circuits over an arbitrary fixed finite basis, where $C_i$ is a circuit of size $O(\log n)$ in the variables $X_i(A^{(m,n)})$ that computes the function $\Sigma_i(A^{(m,n)}, b_i)$. Then every PCR refutation of $\tau(A^{(m,n)}, \vec{C})$ over an arbitrary field with $char(F) \neq 2$ must have size $\exp(\Omega(\frac{n^2}{m(\log n)^3}))$.*
2. *Every PCR refutation of $\tau_{\oplus}(A^{(m,n)}, b)$ over an arbitrary field with $char(F) \neq 2$ must have size $\exp(\Omega(\frac{n^2}{m(\log n)^2}))$.*

*Proof.* The proof is the same as that of Corollary 5.2, only this time we let $s = \log_2 n$. Namely, Theorem 5.1 provides us with an $(r, s, \frac{3}{4}s)$-expander for $r \geq \Omega(n/\log n)$. The proof now follows by Corollary 3.6 and Theorem 4.1. ◻

Corollary 5.3 allows us to construct generators that stretch $n$ bits to $m = o(n^2/(\log n)^4)$ bits in the circuit encoding, and to $m = o(n^2/(\log n)^3)$ bits in linear encoding, which are hard for polynomial size PCR proofs in odd characteristic.

## 6. Recent developments.

Since the preliminary version of this paper (see Report TR00-23 of the Electronic Colloquium on Computational Complexity, and the Proceedings of the 41st IEEE Symposium on Foundations of Computer Science) was disseminated, many open problems presented there have been solved, and many other related developments have occurred.

Alekhnovich and Razborov [AR01] extended our lower bounds for the PC degree (Theorems 3.10 and 3.7) to a large natural class of base functions $g_1, \ldots, g_m$. This class is defined by the requirement that the ideal spanned by every individual $g_i$ does not contain any nonzero multilinear polynomials of low degree.

The principles studied in this paper expressing that Nisan–Wigderson generators are not onto bear a striking similarity to the pigeonhole principle $PHP_n^m$ (with the same meaning of the parameters $m, n$). At the time this paper was written, one of the most interesting open problems, both for NW-generators and for $PHP_n^m$, was to break through the quadratic barrier $m \geq n^2$ for (at least) the resolution size. This has been solved in both contexts.

The pigeonhole principle $PHP_n^m$ was the first to yield. Raz [RanRaz02] proved exponential lower bounds on the size of its resolution refutations when $m \gg n$. Razborov [Raz02a] gave a simpler proof of a somewhat better bound that also holds for the more general functional onto version of this principle.

The quadratic barrier for pseudorandom generators did not stand for much longer. Razborov [Raz02b] constructed Nisan generators (that is, when the base functions $g_i$ are $\mathbb{F}_2$-linear forms) that allow $m \geq n^{\Omega(\log n)}$ output bits and are exponentially hard not only for resolution but also for its extensions $\mathrm{Res}(\epsilon \log n)$ (operating with $(\epsilon \log n)$-DNF instead of clauses) and PCR when $char(F) \neq 2$.

Another question asked in the earlier version of our paper was whether any structural theory of pseudorandom generators is possible within the framework of proof complexity. In particular, we asked whether it is possible to formulate and prove any reasonable statement that would say, possibly in a restricted way, that the composition of hard generators is hard (for a given propositional proof system). This was satisfactorily answered by Krajíček [Kra02], who showed that this is indeed the case, provided hardness is replaced by a stronger notion of $s$-iterability (inspired by the so-called counterexample interpretation).

It was also conjectured in the earlier version that such a composition result might provide an alternate approach to the quadratic barrier problem (but for more complicated generators). This has indeed turned out to be the case. Krajíček [Kra02] proved (independently of [Raz02b]) that our generator from section 4 can be iterated with itself once, which immediately allowed him to get as many as $m = n^{3-\epsilon}$ output bits. The Nisan generator from [Raz02b] turned out to be particularly suitable for Krajíček's notion of $s$-iterability, and it can be composed with itself exponentially many times while preserving hardness. In this way [Raz02b] constructed a function generator with $m = 2^{n^\epsilon}$ outputs which is hard for $\mathrm{Res}(\epsilon \log n)$ and for PCR with $char(F) \neq 2$. Along the lines described in the discussion after Example 3, this immediately implied that neither of these systems possess efficient proofs of $\mathbf{NP} \not\subseteq \mathbf{P}/poly$ (the same conclusion for resolution had already followed from [RanRaz02, Raz02a]).

Finally, [CRVW02] took an important step toward constructing explicit expanders (called there and in [Raz02b] "lossless") with very good expansion properties (even if not sufficient yet for many of our purposes).

## 7. Open problems.

As indicated in the previous section, the most intriguing open problems asked by us in earlier versions have been solved. Some of them, however, remain open.

Can we reduce the devastating $2^{2^s}$ factor in our size lower bounds for the functional framework (Corollaries 3.5 and 3.11)? One way to approach this would be to look for generalizations of the basic Proposition 2.6 that would take into account the structure of the variables $y_f$ (which can be originally divided into $m$ large groups).

Find explicit constructions of $(r, s, c)$-expanders with parameters that would be sufficient for (at least some of) the applications in the current paper and in [Raz02b] (as we remarked above, one step in this direction was made in [CRVW02]).

The bound from [AR01] on the PC degree mentioned in the previous section is not entirely satisfactory since for this bound we need rather good expanders with the expansion ratio $c > 3s/4$. Can we improve it in such a way that it will work under less restrictive conditions, such as similar bounds in Theorems 3.1, 3.7, and 3.10?

More open problems representing the next generation of tasks faced by this theory can be found in [Raz02b].

**Acknowledgment.** We are grateful to both anonymous referees for many useful remarks.

## REFERENCES

[ABRW02] M. Alekhnovich, E. Ben-Sasson, A. A. Razborov, and A. Wigderson, *Space complexity in propositional calculus*, SIAM J. Comput., 31 (2002), pp. 1184–1211.

[Ajt83] M. Ajtai, $\Sigma_1^1$-*formulae on finite structures*, Ann. Pure Appl. Logic, 24 (1983), pp. 1–48.

[AR01] M. Alekhnovich and A. Razborov, *Lower bounds for polynomial calculus: Non-binomial case*, Proc. Steklov Inst. Math., 242 (2003), pp. 18–35.

[Alo98] N. Alon, *Spectral techniques in graph algorithms*, in LATIN'98: Theoretical Informatics (Campinas, 1998), Lecture Notes in Comput. Sci. 1380, C. L. Lucchesi and A. V. Moura, eds., Springer-Verlag, Berlin, 1998, pp. 206–215.

[BFNW93] L. Babai, L. Fortnow, N. Nisan, and A. Wigderson, *BPP has subexponential time simulations unless EXPTIME has publishable proofs*, Complexity, 3 (1993), pp. 307–318.

[BP96] P. Beame and T. Pitassi, *Simplified and improved resolution lower bounds*, in Proceedings of the 37th IEEE Symposium on Foundations of Computer Science, 1996, pp. 274–282.

[BP98] P. Beame and T. Pitassi, *Propositional proof complexity: Past, Present, and Future*, Bull. Eur. Assoc. Theor. Comput. Sci. EATCS, 65 (1998), pp. 66–89.

[BST98] P. Beame, M. Saks, and J. S. Thathachar, *Time-space tradeoffs for branching programs*, in Proceedings of the 39th IEEE Symposium on Foundations of Computer Science, 1998, pp. 254–263.

[BI99] E. Ben-Sasson and R. Impagliazzo, *Random CNF's are hard for the polynomial calculus*, in Proceedings of the 40th IEEE Symposium on Foundations of Computer Science, 1999, pp. 415–421.

[BW99] E. Ben-Sasson and A. Wigderson, *Short proofs are narrow—resolution made simple*, in Proceedings of the 31st ACM Symposium on Theory of Computing, 1999, pp. 517–526.

[BG99] M. Bonet and N. Galesi, *A study of proof search algorithms for resolution and polynomial calculus*, in Proceedings of the 40th IEEE Symposium on Foundations of Computer Science, 1999, pp. 422–432.

[BGIP01] S. Buss, D. Grigoriev, R. Impagliazzo, and T. Pitassi, *Linear gaps between degrees for the polynomial calculus modulo distinct primes*, J. Comput. System Sci., 62 (2001), pp. 267–289.

[CEI96] M. Clegg, J. Edmonds, and R. Impagliazzo, *Using the Groebner basis algorithm to find proofs of unsatisfiability*, in Proceedings of the 28th ACM Symposium on Theory of Computing, 1996, pp. 174–183.

[CRVW02] M. Capalbo, O. Reingold, S. Vadhan, and A. Wigderson, *Randomness conductors and constant-degree expansion beyond the degree/2 barrier*, in Proceedings of the 34th ACM Symposium on the Theory of Computing, 2002, pp. 659–668.

[FSS84] M. Furst, J. B. Saxe, and M. Sipser, *Parity, circuits and the polynomial time hierarchy*, Math. Systems Theory, 17 (1984), pp. 13–27.

[Gri98] D. Grigoriev, *Tseitin's tautologies and lower bounds for Nullstellensatz proofs*, in Proceedings of the 39th IEEE Symposium on Foundations of Computer Science, 1998, pp. 648–652.

[Gri01] D. Grigoriev, *Linear lower bounds on degrees of Postivestellensatz calculus proofs for the parity*, Theoret. Comput. Sci., 259 (2001), pp. 613–622.

[GZ97] O. Goldreich and D. Zuckerman, *Another Proof that BPP⊆PH (and More)*, Technical Report TR97-045, Electronic Colloquium on Computational Complexity, 1997.

[Hak85] A. Haken, *The intractability or resolution*, Theoret. Comput. Sci., 39 (1985), pp. 297–308.

[Hås86] J. Håstad, *Computational Limitations on Small Depth Circuits*, Ph.D. thesis, MIT, Cambridge, MA, 1986.

[IKW01] R. Impagliazzo, V. Kabanets, and A. Wigderson, *In search for an easy witness: Exponential time vs. probabilistic polynomial time*, in Proceedings of the 16th Annual IEEE Conference on Computational Complexity, 2001, pp. 2–12.

[IPS99] R. Impagliazzo, P. Pudlák, and J. Sgall, *Lower bounds for the polynomial calculus and the Groebner basis algorithm*, Comput. Complexity, 8 (1999), pp. 127–144.

[IW97] R. Impagliazzo and A. Wigderson, *$P = BPP$ if E requires exponential circuits: Derandomizing the XOR Lemma*, in Proceedings of the 29th Annual ACM Symposium on Theory of Computing, 1997, pp. 220–229.

[Kra97] J. Krajíček, *Interpolation theorems, lower bounds for proof systems, and independence results for bounded arithmetic*, J. Symbolic Logic, 62 (1997), pp. 457–486.

[Kra01] J. Krajíček, *On the weak pigeonhole principle*, Fund. Math., 170 (2001), pp. 123–140.

[Kra02] J. Krajíček, *Dual weak pigeonhole principle, pseudo-surjective functions, and provability of circuit lower bounds*, J. Symbolic Logic, 69 (2004), pp. 265–286.

[N91] N. Nisan, *Pseudo-random bits for constant-depth circuits*, Combinatorica, 11 (1991), pp. 63–70.

[NW94] N. Nisan and A. Wigderson, *Hardness vs. randomness*, J. Comput. System Sci., 49 (1994), pp. 149–167.

[RanRaz02] R. Raz, *Resolution lower bounds for the weak pigeonhole principle*, in Proceedings of the 34th ACM Symposium on the Theory of Computing, 2002, pp. 553–562.

[Raz95a] A. Razborov, *Bounded arithmetic and lower bounds in Boolean complexity*, in Feasible Mathematics II, P. Clote and J. Remmel, eds., Progr. Comput. Sci. Appl. Logic 13, Birkhäuser Boston, Boston, 1995, pp. 344–386.

[Raz95b] A. Razborov, *Unprovability of lower bounds on the circuit size in certain fragments of bounded arithmetic*, Izv. Ross. Akad. Nauk Ser. Mat., 59 (1995), pp. 201–224.

[Raz96] A. Razborov, *Lower bounds for propositional proofs and independence results in bounded arithmetic*, in Proceedings of the 23rd International Colloquium on Automata, Languages, and Programming, F. Meyer auf der Heide and B. Monien, eds., Lecture Notes in Comput. Sci. 1099, Springer-Verlag, New York, Berlin, 1996, pp. 48–62.

[Raz98] A. Razborov, *Lower bounds for the polynomial calculus*, Comput. Complexity, 7 (1998), pp. 291–324.

[Raz02a] A. Razborov, *Resolution lower bounds for perfect matching principles*, in Proceedings of the 17th IEEE Conference on Computational Complexity, 2002, pp. 29–38.

[Raz02b] A. Razborov, *Pseudorandom Generators Hard for k-DNF Resolution and Polynomial Calculus Resolution*, manuscript; available online from http://www.genesis.mi.ras.ru/~razborov, 2002.

[RR97] A. Razborov and S. Rudich, *Natural proofs*, J. Comput. System Sci., 55 (1997), pp. 24–35.

[Tse68] G. C. Tseitin, *On the complexity of derivations in propositional calculus*, in Studies in Mathematics and Mathematical Logic, Part II, A. O. Slissenko, ed., Consultants Bureau, New York, London, 1968, pp. 115–125.

[Urq87] A. Urquhart, *Hard examples for resolution*, J. ACM, 34 (1987), pp. 209–219.

[Yao82] A. Yao, *Theory and applications of trapdoor functions*, in Proceedings of the 23rd IEEE Symposium on Foundations of Computer Science, 1982, pp. 92–99.

[Yao85] A. Yao, *Separating the polynomial-time hierarchy by oracles*, in Proceedings of the 26th Symposium on Foundations of Computer Science, 1985, pp. 1–10.

SIAM J. Comput.
Vol. 34, No. 1, pp. 89–108

# ALGORITHMS FOR RH MAPPING: NEW IDEAS AND IMPROVED ANALYSIS*

LARS IVANSSON† AND JENS LAGERGREN†

**Abstract.** Radiation hybrid (RH) mapping is a technique for constructing a physical map describing the locations of $n$ markers on a chromosome of an organism. In [*J. Comput. Biol.*, 4 (1997), pp. 517–533], Ben-Dor and Chor presented new algorithms for the RH problem and gave the first performance guarantees for such algorithms. We improve the lower bounds on the number of experiments in a way that is sufficient for two of these algorithms to give a correct ordering of the markers with high probability. Not only are the new bounds tighter, but our analysis also captures to a much higher extent how the bounds depend on the actual arrangement of the markers. Furthermore, we modify the two algorithms to utilize RH mapping data produced with several radiation intensities. We show that the new algorithms are almost insensitive to the problem of using the correct intensity.

**Key words.** RH mapping, algorithms, performance bounds, multiple intensities

**AMS subject classifications.** 68Q25, 68W40

**DOI.** 10.1137/S0097539701388355

## 1. Introduction.

Physical mapping is an important problem in large-scale sequencing of DNA as well as for locating genes. In RH (radiation hybrid) mapping, a physical map describing the locations of $n$ markers on a chromosome of an organism is constructed. The markers can be genes or arbitrary DNA sequences, and the resulting information consists of the relative order of these markers and the distance between them on the chromosome. The RH mapping procedure can be divided into an experimental and an algorithmic part. In the experimental part a series of RH experiments is made, which yields pairwise distances between markers. In the algorithmic part an algorithm generates a map of the markers, given these distances. Here, we study the latter algorithmic part, called the RH problem.

There exist several algorithms for the RH problem. Most algorithms are heuristics [4, 10, 13], but for a few algorithms theoretical results exist as well. In [2], for instance, Ben-Dor and Chor presented three algorithms for the RH problem, together with bounds on the number of experiments that is sufficient for the algorithms to give the correct order of the markers in the map. In [7], an approximation algorithm for the Matrix-To-Line problem was used to generate a map from pairwise distances. Arrangements produced with this algorithm were shown to converge to the true arrangements.

The three algorithms in [2] are called `P-Order`, `K-Order`, and `MST-Order`. The traveling salesman problem (TSP)-based algorithm `MST-Order` was developed into the program `RHO` [3], which has been even further developed using the combinatorial optimization package CONCORDE [1]. TSP/CONCORDE has been compared to other RH mapping software [8], and has been practically used to create RH maps for the canine genome [6] as well as the feline genome [12]. In this paper we improve the analysis of the other two algorithms: `P-Order` and `K-Order`, inspired by Prim's and Kruskal's algorithms (see [5]), respectively. We obtain better lower bounds on the

*Received by the editors April 24, 2001; accepted for publication (in revised form) May 4, 2004; published electronically October 8, 2004.
http://www.siam.org/journals/sicomp/34-1/38835.html

†Stockholm Bioinformatics Center, Department of Numerical Analysis and Computer Science, KTH, Stockholm, Sweden (ivan@nada.kth.se, jensl@nada.kth.se).

number of experiments sufficient to obtain the correct order of the markers with high probability. Furthermore, our bounds capture the impact that the true arrangement of the markers has on the performance of the two algorithms. In fact, it is possible to construct instances for which the gap between our bound and the previous known bounds are arbitrarily large. Our analysis also allows us to gain insight into the question of for which arrangements the algorithms are likely to perform well and for which arrangements they are more likely to fail. The improved analysis suggests that the `K-Order` algorithm is the better of the two algorithms in most cases.

The new analysis of the `P-Order` and `K-Order` algorithms suggests that the use of several intensities for the radiation in the RH experiments could improve the performance bounds for the two algorithms. We propose modified versions of the `P-Order` and `K-Order` algorithms, which allow for the use of RH data produced with several intensities. We show that, given $O(\log n)$ series of experiments, using a range of intensities, the dependency on the choice of intensity can be removed from the performance bounds, under the assumption that the markers are uniformly distributed along the chromosome. Although multiple intensity algorithms have been proposed before [11], this is, to our knowledge, the first performance bound for such an algorithm.

## 2. The probabilistic model.

A marker is a gene or an arbitrary DNA sequence the presence of which can be detected in any DNA fragment via a laboratory test. To obtain information about the order of and distance between a pair of markers on a fragment of DNA, an RH experiment is performed. In an RH experiment the chromosome is exposed to gamma radiation, which shatters it into fragments. Some of the fragments are incorporated into (retained in) a hamster cell, which is grown to yield a hybrid cell line. Cells from this cell line are then tested for the presence of each DNA marker. The outcome of one experiment is represented by a vector in $\{0,1\}^n$, where 1 in the $i$th position corresponds to the presence of the $i$th marker; the outcome from $m$ experiments is in the natural way represented by an $m \times n$-matrix consisting of 0's and 1's.

The probabilistic model as well as the notation used in this paper are the same as, for instance, those used in [2] and [7]. We let $\lambda$ be the intensity of the gamma radiation used in the RH experiment. We let $p$ be the retention probability, i.e., the probability that a fragment is incorporated into the hamster cell. To simplify the calculations in section 4, we assume that this probability is independent of the radiation intensity. In practice, the retention probability goes down when the intensity goes up. In testing for the presence of a marker, we allow for errors. We let $\alpha$ be the probability of a false positive answer, and $\beta$ be the probability of a false negative answer. Furthermore, we assume that the retention of the different fragments, as well as the tests for presence, are independent events.

In [7] it was shown that, under this model, the separation probability for two markers $a$ and $b$ is

$$\varphi_{a,b} = 2pq(1 - e^{-\lambda \ell_{ab}})(1 - (\alpha + \beta))^2 + g(\alpha, \beta, p), \tag{2.1}$$

where $q = 1 - p$, $\ell_{ab}$ is the distance between $a$ and $b$, and

$$g(\alpha, \beta, p) = 2p(\alpha - \beta)(\alpha + \beta - 1) + 2\alpha(1 - \alpha). \tag{2.2}$$

The separation probabilities can be estimated from a series of RH experiments. With

$S_i(a,b)$ as the random variable defined by

$$S_i(a,b) = \begin{cases} 1 & \text{if } a \text{ and } b \text{ are separated in experiment } i, \\ 0 & \text{otherwise,} \end{cases} \tag{2.3}$$

for $i = 1, \ldots, m$, we get an estimate $\hat{\varphi}_{a,b}$ through

$$\hat{\varphi}_{a,b} = \frac{1}{m} \sum_{i=1}^{m} S_i(a,b). \tag{2.4}$$

We note that this estimate is unbiased since $E[\hat{\varphi}_{a,b}] = \varphi_{a,b}$. From (2.1), an expression can be obtained for the distance between two markers as a function of the separation probability. Using the estimates $\hat{\varphi}_{a,b}$ in this expression, we obtain estimates $\hat{d}(a,b)$ of the physical distances between any two markers $a$ and $b$, i.e.,

$$\hat{d}(a,b) = -\frac{1}{\lambda} \ln \left( 1 - \frac{\hat{\varphi}_{a,b} - g(\alpha, \beta, p)}{2pq(1-(\alpha+\beta))^2} \right). \tag{2.5}$$

However, we are not as interested in the estimated distance between two markers $a$ and $b$ as in whether the estimated distance between them is greater than the estimated distance between two other markers $c$ and $d$. Since the separation probability is an increasing function of the distance, this is equivalent to the question of whether $\hat{\varphi}_{a,b} > \hat{\varphi}_{c,d}$. More precisely, we are interested in the probability that this inequality holds, given that $d(a,b) > d(c,d)$. In this context it is natural to define a *distance comparator.*

DEFINITION 2.1. *A function $\mathcal{C}(a,b;c,d)$ from quadruples of markers to the set of integers $\{-1,0,1\}$ is a distance comparator if, for all choices of markers $a$, $b$, $c$, and $d$,*

1. $\mathcal{C}(a,b;c,d) = \mathcal{C}(b,a;c,d)$,
2. $\mathcal{C}(a,b;c,d) = \mathcal{C}(a,b;d,c)$,
3. $\mathcal{C}(a,b;c,d) + \mathcal{C}(c,d;a,b) = 0$.

We should think of a distance comparator as any procedure that compares pairs of distances between markers. If the procedure decides that the distance between two markers $a$ and $b$ is greater than the distance between two other markers $c$ and $d$, then $\mathcal{C}(a,b;c,d) = 1$; if it decides that the distance between $a$ and $b$ is less than the distance between $c$ and $d$, then $\mathcal{C}(a,b;c,d) = -1$; and if it decides that the distances are equal, then $\mathcal{C}(a,b;c,d) = 0$.

The specific distance comparator induced by the distance estimate $\hat{d}(a,b)$ above will be denoted $\widehat{\mathcal{C}}$.

DEFINITION 2.2. *Let $\hat{d}(x,y)$ be the distance estimate defined by* (2.5). *Define*

$$\widehat{\mathcal{C}}(a,b;c,d) = \begin{cases} 1 & \text{if } \hat{d}(a,b) > \hat{d}(c,d), \\ 0 & \text{if } \hat{d}(a,b) = \hat{d}(c,d), \\ -1 & \text{if } \hat{d}(a,b) < \hat{d}(c,d). \end{cases} \tag{2.6}$$

This means that $\widehat{\mathcal{C}}$ is the distance comparator used to compare distances in the `P-Order` and `K-Order` algorithms. Using Definition 2.2, we can rewrite the probability for which we want to find a lower bound as the probability that $\widehat{\mathcal{C}}(a,b;c,d) = 1$, given that $d(a,b) > d(c,d)$.

As was pointed out above, the question of whether $\widehat{\mathcal{C}}(a,b;c,d) = 1$, given that $d(a,b) > d(c,d)$, is equivalent to the question of whether $\hat{\varphi}_{a,b} > \hat{\varphi}_{c,d}$, given that $d(a,b) > d(c,d)$. This question bears close resemblance to the question of whether the estimated separation probabilities are what Ben-Dor and Chor call *consistent* or not [2]. The difference is that to show consistency, the inequality has to be true only for the case when $c$ and $d$ are consecutive in the true order and are located between $a$ and $b$. It turns out, however, that the calculations in [2] can be carried through in this more general case as well.

LEMMA 2.3. *Let $a, b, c,$ and $d$ be four markers such that $d(a,b) > d(c,d)$. Then*

$$\Pr\big[\widehat{\mathcal{C}}(a,b;c,d) \neq 1\big] \leq e^{-2mp^2q^2(1-(\alpha+\beta))^4 f_{\ell_{ab}-\ell_{cd},\ell_{cd}}(\lambda)^2}, \tag{2.7}$$

*where $f_{\ell_1,\ell_2}(\lambda) = (1-e^{-\lambda\ell_1})e^{-\lambda\ell_2}$.*

*Proof.* Let $X_i$, $i = 1,\dots,m$, be the random variables defined by $X_i = 1$ if $a, b$ but not $c, d$ are separated in experiment $i$; $X_i = -1$ if $c, d$ but not $a, b$ are separated in experiment $i$; and $X_i = 0$ otherwise. We observe that if we let

$$Y = \sum_{i=1}^{m} X_i, \tag{2.8}$$

then $\hat{\varphi}_{a,b} - \hat{\varphi}_{c,d} = Y/m$. Hence $Y > 0$ if and only if $\widehat{\mathcal{C}}(a,b;c,d) = 1$. From (2.1) it follows that

$$\begin{aligned} \varphi_{a,b} - \varphi_{c,d} &= 2pq(1-(\alpha+\beta))^2(1-e^{-\lambda(\ell_{ab}-\ell_{cd})})e^{-\lambda\ell_{cd}} \\ &= 2pq(1-(\alpha+\beta))^2 f_{\ell_{ab}-\ell_{cd},\ell_{cd}}(\lambda), \end{aligned} \tag{2.9}$$

which means that the expectation $E[Y]$ can be written

$$\begin{aligned} E[Y] &= m(\varphi_{a,b} - \varphi_{c,d}) \\ &= 2mpq(1-(\alpha+\beta))^2 f_{\ell_{ab}-\ell_{cd},\ell_{cd}}(\lambda). \end{aligned} \tag{2.10}$$

Our aim is to bound the probability that $Y < 0$, given that $E[Y] > 0$, i.e., the probability that $Y$ deviates from $E[Y]$ toward zero by more than $E[Y]$. Since the variables $X_i$ are independent, Hoeffding's inequality [9] immediately yields that

$$\begin{aligned} \Pr\big[E[Y] - Y \geq E[Y]\big] &\leq e^{-2E[Y]^2/4m} \\ &= e^{-2mp^2q^2(1-(\alpha+\beta))^4 f_{\ell_{ab}-\ell_{cd},\ell_{cd}}(\lambda)^2}. \end{aligned} \tag{2.11}$$ ∎

The function $f_{\ell_1,\ell_2}(\lambda)$ can be given the following interpretation. If $I_1$ and $I_2$ are two intervals of length $\ell_1$ and $\ell_2$, respectively, then

$$\begin{aligned} &\Pr\big[(\text{at least one break occurs in } I_1) \wedge (\text{no breaks occur in } I_2)\big] \\ &\quad = (1-e^{-\lambda\ell_1})e^{-\lambda\ell_2} = f_{\ell_1,\ell_2}(\lambda). \end{aligned} \tag{2.12}$$

This function will be explored in more detail later, but for now the following intuitively obvious properties will be sufficient to carry through our derivations.

LEMMA 2.4. *Let $\lambda > 0$. If $\ell \geq \ell_1 > 0$, then $f_{\ell,\ell_2}(\lambda) \geq f_{\ell_1,\ell_2}(\lambda)$. If $\ell \geq \ell_2 > 0$, then $f_{\ell_1,\ell}(\lambda) \leq f_{\ell_1,\ell_2}(\lambda)$. If $\ell_1 \geq \ell_2 > 0$, then $f_{\ell_1,\ell_2}(\lambda) \geq f_{\ell_2,\ell_1}(\lambda)$.*

*Proof.* Assume that $\lambda > 0$. If $\ell \geq \ell_1 \geq 0$, then

$$f_{\ell,\ell_2}(\lambda) = (1-e^{-\lambda\ell})e^{-\lambda\ell_2} \geq (1-e^{-\lambda\ell_1})e^{-\lambda\ell_2} = f_{\ell_1,\ell_2}(\lambda). \tag{2.13}$$

If $\ell \geq \ell_2 \geq 0$, then

$$f_{\ell_1,\ell}(\lambda) = (1 - e^{-\lambda\ell_1})e^{-\lambda\ell} \leq (1 - e^{-\lambda\ell_1})e^{-\lambda\ell_2} = f_{\ell_1,\ell_2}(\lambda). \tag{2.14}$$

If $\ell_1 \geq \ell_2 \geq 0$, then

$$f_{\ell_1,\ell_2}(\lambda) = (1 - e^{-\lambda\ell_1})e^{-\lambda\ell_2} \geq (1 - e^{-\lambda\ell_2})e^{-\lambda\ell_1} = f_{\ell_2,\ell_1}(\lambda). \qquad \square \tag{2.15}$$

## 3. An improved analysis of known algorithms.

In [2], Ben-Dor and Chor proposed two simple algorithms for the calculation of the marker order: the `P-Order` and `K-Order` algorithms. In the `P-Order` algorithm a chain of markers is built in a stepwise fashion. Initially, a chain is constructed by connecting the two markers with the shortest estimated distance. Then, in each step of the algorithm, the marker with the shortest estimated distance to any of the two ends of the chain is connected to that end of the chain. In the `K-Order` algorithm, a collection of chains is maintained. Initially there are $n$ chains of length 1. Then, in each step of the algorithm, the two endmarkers with the shortest estimated distance between them are connected.

It was shown in [2] that if $\delta_{\max}$ and $\delta_{\min}$ are the longest and shortest, respectively, distances between any pair of consecutive markers on the chromosome, then given

$$m \geq \log\left(\frac{n^3}{\epsilon}\right) \frac{1}{2p^2q^2(1-(\alpha+\beta))^4 f_{\delta_{\min},\delta_{\max}}(\lambda)^2} \tag{3.1}$$

experiments, the estimated distances are *consistent* with probability $1-\epsilon$, i.e., that $\hat{d}(a,d) > \hat{d}(b,c)$ for *all* markers $a$, $b$, $c$, and $d$, appearing in this order on the genome, such that $b$ and $c$ are consecutive and $d(a,d) > d(b,c)$. Furthermore, they showed that if the estimated distances are consistent, both the `P-Order` and the `K-Order` algorithms yield a correct order of the markers. We prove new tighter bounds for both the `P-Order` and the `K-Order` algorithms. These new bounds improve the above bound, by taking into account the actual arrangement of the markers.

### 3.1. Analyzing the `P-Order` algorithm.

In the `P-Order` algorithm one marker at a time is added to the chain that is being built. In order to show correctness of the `P-Order` algorithm we thus need to assert that, with high probability, a marker adjacent to an endmarker of the chain is added to that marker in each step, i.e., that the estimated distance between the endmarker and the marker adjacent to that marker is smaller than the estimated distance between the endmarker and any other marker. The following definition will be useful for this purpose.

DEFINITION 3.1. *A distance comparator $\mathcal{C}$ satisfies the* `P-Order` *condition with parameter $\gamma$ if, for any markers $a$, $b$, $c$, and $d$ appearing in this order on the chromosome such that $b$ and $c$ are consecutive and $d(a,d) > d(b,c)$, the probability that $\mathcal{C}(a,d;b,c) = 1$ is at least $1-\gamma$.*

Note the difference between the concept of consistency and the `P-Order` condition. For a distance estimate to be consistent it is required that $\hat{d}(b,c) \leq \hat{d}(a,d)$ for *all* markers $a$, $b$, $c$, and $d$ appearing in this order on the chromosome such that $b$ and $c$ are consecutive and $d(a,d) > d(b,c)$. It turns out to be sufficient that the `P-Order` condition hold for $n^2$ such quadruples of markers simultaneously, namely, the pairs of distances actually compared in the algorithm.

THEOREM 3.2. *If a distance comparator $\mathcal{C}$ satisfying the* `P-Order` *condition with parameter $\gamma$ is used in the* `P-Order` *algorithm, then the* `P-Order` *algorithm will give the correct order of the markers with probability greater than $1-\gamma n^2$.*

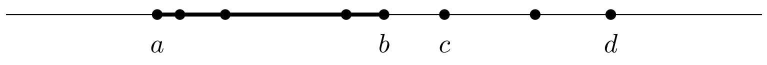


FIG. 3.1. *The kth step of the* `P-Order` *algorithm.*

*Proof.* Let $a$ be an arbitrarily chosen marker, and let $b$ be the marker immediately to the right of $a$. From the `P-Order` condition it follows that $\mathcal{C}(a,b;a,c) = -1$ with probability at least $1-\gamma$ for any marker $c$ to the right of $b$. Since there are at most $n-2$ markers to the right of $b$, this means that, with probability at least $1-\gamma(n-2)$, $\mathcal{C}(a,b;a,c) = -1$ for all markers $c$ to the right of $b$. Let $A_1$ be the event that the first two markers connected in the `P-Order` algorithm are adjacent, and let $L(i)$ be the event that the $i$th marker is the leftmost marker of the two markers chosen. Then

$$\begin{aligned} \Pr\bigl[A_1\bigr] &= \sum_{i=1}^{n-1} \Pr\bigl[A_1 \mid L(i)\bigr] \cdot \Pr\bigl[L(i)\bigr] \\ &\geq \sum_{i=1}^{n-1} (1-\gamma(n-2)) \cdot \Pr\bigl[L(i)\bigr] \\ &= 1-\gamma(n-2). \end{aligned} \tag{3.2}$$

Hence, the first two markers connected in the `P-Order` algorithm are adjacent with probability $\geq 1-\gamma(n-2)$.

Now, assume that the `P-Order` algorithm has correctly constructed a chain consisting of $k$ markers ($k \geq 2$). Let $a$ be the left endpoint of the chain, and let $b$ be the right endpoint of the chain. Finally, let $c$ be the marker immediately to the right of $b$; see Figure 3.1. From the `P-Order` condition it follows that $\mathcal{C}(b,c;b,d) = -1$ with probability $\geq 1-\gamma$ for any marker $d$ to the right of $c$. Furthermore, it follows that $\mathcal{C}(b,c;a,d) = -1$ with probability $\geq 1-\gamma$ for any marker $d$ to the right of $b$. The same derivations can obviously be carried through for the markers to the left of $a$.

In the $k$th step of the algorithm ($k \geq 2$) there are $n-k$ nonconnected markers. Let $A_k$ be the event that a marker adjacent to one of the endmarkers of the chain is connected to that endmarker in step $k$. Let $L$ be the event that the new marker is added to the left endmarker of the chain, and let $R$ be the event that the new marker is added to the right endmarker of the chain. Then

$$\begin{aligned} \Pr\bigl[A_k \mid A_1,\dots,A_{k-1}\bigr] &= \Pr\bigl[A_k \mid L, A_1,\dots,A_{k-1}\bigr] \cdot \Pr\bigl[L \mid A_1,\dots,A_{k-1}\bigr] \\ &\quad + \Pr\bigl[A_k \mid R, A_1,\dots,A_{k-1}\bigr] \cdot \Pr\bigl[R \mid A_1,\dots,A_{k-1}\bigr] \\ &\geq (1-2\gamma(n-k)) \cdot \Pr\bigl[L \mid A_1,\dots,A_{k-1}\bigr] \\ &\quad + (1-2\gamma(n-k)) \cdot \Pr\bigl[R \mid A_1,\dots,A_{k-1}\bigr] \\ &= 1-2\gamma(n-k). \end{aligned} \tag{3.3}$$

We thus conclude that if only adjacent markers have been connected in the first $k-1$ steps of the `P-Order` algorithm, then the marker being connected to an endmarker in the $k$th step is adjacent to that marker with probability $\geq 1-2\gamma(n-k)$.

If we sum up all steps of the algorithm, including the first step, we get an upper bound on the error probability $\epsilon$:

$$\epsilon \leq \gamma(n-2) + \sum_{k=2}^{n} 2\gamma(n-k) < \sum_{j=1}^{n-1} 2\gamma j = \gamma n(n-1) \leq \gamma n^2. \tag{3.4}$$

We have thus shown that with probability greater than $1-\gamma n^2$ the `P-Order` algorithm gives the correct order of the markers. ◻

Now, we will show how to find an expression for the parameter in the `P-Order` condition such that the distance comparator $\widehat{\mathcal{C}}$ satisfies the `P-Order` condition. This expression will obviously be heavily dependent on how the markers are arranged on the chromosome, as well as on the radiation intensity used in the experiment.

DEFINITION 3.3. *Define the function* $f_{\mathtt{P}}(\lambda)$ *as*

$$f_{\mathtt{P}}(\lambda) = \min_{1<i<n} \{ f_{\ell_{i-1,i},\ell_{i,i+1}}(\lambda), f_{\ell_{i,i+1},\ell_{i-1,i}}(\lambda) \}. \tag{3.5}$$

LEMMA 3.4. *Given* $m$ *experiments, the distance comparator* $\widehat{\mathcal{C}}$*, defined in Definition* 2.2*, satisfies the* `P-Order` *condition with parameter*

$$e^{-2mp^2q^2(1-(\alpha+\beta))^4 f_{\mathtt{P}}(\lambda)^2}. \tag{3.6}$$

*Proof.* Let $a$, $b$, $c$, and $d$ be four markers appearing in this order on the chromosome; assume that $b$ and $c$ are consecutive and that $d(a,d) > d(b,c)$. Without loss of generality we assume that $d \neq c$. Let $e$ be the marker immediately to the right of $c$. Note that $e$ may be identical to the marker $d$. Now, it follows from Lemmas 2.3 and 2.4 that

$$\begin{aligned} \Pr\big[\widehat{\mathcal{C}}(a,d;b,c) \neq 1\big] &\leq e^{-2mp^2q^2(1-(\alpha+\beta))^4 f_{\ell_{ad}-\ell_{bc},\ell_{bc}}(\lambda)^2} \\ &= e^{-2mp^2q^2(1-(\alpha+\beta))^4 f_{\ell_{ce},\ell_{bc}}(\lambda)^2} \\ &\leq e^{-2mp^2q^2(1-(\alpha+\beta))^4 f_{\mathtt{P}}(\lambda)^2}, \end{aligned} \tag{3.7}$$

where the last inequality follows from the fact that the markers $b$, $c$, $e$ are consecutive, together with Definition 3.3. Hence, for any four markers $a$, $b$, $c$, and $d$ appearing in this order on the chromosome such that $b$ and $c$ are consecutive and $d(a,d) > d(b,c)$,

$$\Pr\big[\widehat{\mathcal{C}}(a,d;b,c) = 1\big] \geq 1 - e^{-2mp^2q^2(1-(\alpha+\beta))^4 f_{\mathtt{P}}(\lambda)^2}. \tag{3.8}$$ ◻

If we use Lemma 3.4 in combination with Theorem 3.2, we get the following theorem, which gives an upper bound on the number of experiments that is sufficient for the `P-Order` algorithm to give the correct order of the markers.

THEOREM 3.5. *Given*

$$m \geq \ln\left(\frac{n^2}{\epsilon}\right) \frac{1}{2p^2q^2(1-(\alpha+\beta))^4 f_{\mathtt{P}}(\lambda)^2} \tag{3.9}$$

*experiments, the* `P-Order` *algorithm gives the correct order of the markers with probability greater than* $1-\epsilon$.

*Proof.* From Lemma 3.4 it follows that with

$$m \geq \ln\left(\frac{n^2}{\epsilon}\right) \frac{1}{2p^2q^2(1-(\alpha+\beta))^4 f_{\mathtt{P}}(\lambda)^2} \tag{3.10}$$

experiments, $\widehat{\mathcal{C}}$ satisfies the `P-Order` condition with parameter $\leq \epsilon/n^2$. By Theorem 3.2, this implies that the `P-Order` algorithm will give the correct order of the markers with probability greater than $1-\epsilon$. ◻

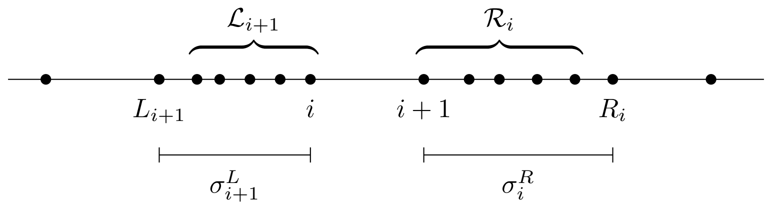


FIG. 3.2. *Notation used in the analysis of the* `K-Order` *algorithm.*

**3.2. Analyzing the `K-Order` algorithm.** In the `K-Order` algorithm the two markers with the shortest estimated distance are joined in each step. This suggests that sequences of tightly packed markers surrounded by long distances probably will be connected to each other before any of the markers are connected to surrounding markers that are far away. This will greatly facilitate the objective of correctly ordering the tightly packed markers, since in this case we just have to order the distances between the surrounding markers and the ends of the chain, instead of the distances between the surrounding markers and the individual markers in the cluster. In order to analyze the algorithm rigorously we will use the following notation (for an illustration, see Figure 3.2).

DEFINITION 3.6.

$$L_i = \max\left(\left\{k : k < i, \ell_{k-1,k} > \frac{\ell_{i-1,i}}{2}\right\} \cup \{1\}\right), \tag{3.11}$$

$$R_i = \min\left(\left\{k : k > i, \ell_{k,k+1} > \frac{\ell_{i,i+1}}{2}\right\} \cup \{n\}\right), \tag{3.12}$$

$$\mathcal{L}_i = \{j : L_i < j \leq i-1\}, \qquad \mathcal{R}_i = \{j : i+1 \leq j < R_i\}, \tag{3.13}$$

$$m_i = \max(\{\ell_{j-1,j} : j \in \mathcal{L}_i\} \cup \{\ell_{j,j+1} : j \in \mathcal{R}_i\}), \tag{3.14}$$

$$\sigma^L_i = \ell_{i-2,i-1} + \sum_{j=L_i+1}^{i-2} \ell_{j-1,j}, \qquad \sigma^R_i = \ell_{i+1,i+2} + \sum_{j=i+2}^{R_i-1} \ell_{j,j+1}. \tag{3.15}$$

The idea is thus to make sure that before we connect the markers $i$ and $i+1$, all short edges to the left of $i$ and to the right of $i+1$ are connected. To capture this idea we define what we call the `K-Order` conditions, which have two parameters.

DEFINITION 3.7. *A distance comparator $\mathcal{C}$ satisfies the* `K-Order` *conditions with parameters $\gamma$ and $\zeta$ if*

1. *for any markers $a$, $b$, $c$, and $d$ appearing in this order on the chromosome such that $b$ and $c$ are consecutive, $d(a,d) > d(b,c)$, and $d \notin \mathcal{R}_b$ or $a \notin \mathcal{L}_c$, the probability that $\mathcal{C}(a,d;b,c) = 1$ is at least $1-\gamma$;*
2. *for each $a \in [2,n]$ and $b \in \mathcal{L}_a$ the probability that $\mathcal{C}(a-1,a;b-1,b) = 1$ is at least $1-\zeta$;*
3. *for each $a \in [1,n-1]$ and $b \in \mathcal{R}_a$ the probability that $\mathcal{C}(a,a+1;b,b+1) = 1$ is at least $1-\zeta$.*

It turns out that if a distance comparator satisfies these `K-Order` conditions, then it is possible to show that the `K-Order` algorithm using this distance comparator will find the true order of the markers with high probability.

THEOREM 3.8. *If a distance comparator $\mathcal{C}$ satisfying the* `K-Order` *condition with parameters $\gamma$ and $\zeta$ is used in the* `K-Order` *algorithm, then the* `K-Order` *algorithm will*

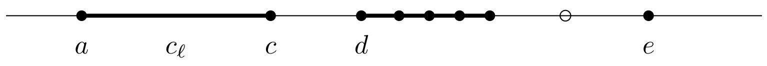


FIG. 3.3. *A step in the* `K-Order` *algorithm.*

*give the correct order of the markers with probability greater than* $1 - 2\gamma n^2 - \zeta n^2$.

*Proof.* We say that a distance comparator $\mathcal{C}$ is $\mathcal{L}$*-closed* if, for all $a \in [2, n]$ and $b \in \mathcal{L}_a$, we have $\mathcal{C}(a-1, a; b-1, b) = 1$; analogously, we say that $\mathcal{C}$ is $\mathcal{R}$*-closed* if, for all $a \in [1, n-1]$ and $b \in \mathcal{R}_a$, we have $\mathcal{C}(a, a+1; b, b+1) = 1$. This means that if $\mathcal{C}$ is $\mathcal{L}$-closed and if the `K-Order` algorithm has only connected adjacent markers, then all markers in $\mathcal{L}_a$ will be connected before the markers $a-1$ and $a$ are connected to each other. Analogously, if $\mathcal{C}$ is $\mathcal{R}$-closed and if the `K-Order` algorithm has only connected adjacent markers, then all markers in $\mathcal{R}_a$ will be connected before the markers $a$ and $a+1$ are connected to each other.

To prove the theorem, we will therefore show that if the distance comparator $\mathcal{C}$, used in the `K-Order` algorithm, satisfies the `K-Order` conditions with parameters $\gamma$ and $\zeta$, then

1. $\mathcal{C}$ is $\mathcal{L}$-closed and $\mathcal{R}$-closed, with probability at least $1 - \zeta n^2$;
2. if $\mathcal{C}$ is $\mathcal{L}$-closed and $\mathcal{R}$-closed, then only adjacent markers will be connected by the `K-Order` algorithm with probability at least $1 - 2\gamma n^2$.

From the `K-Order` conditions it follows that for any marker $a$ and any marker $b \in \mathcal{L}_a$, $\mathcal{C}(a-1, a; b-1, b) = 1$ with probability at least $1 - \zeta$. There are at most $\binom{n}{2}$ pairs of markers altogether, so with probability at least $1 - \zeta n^2/2$, $\mathcal{C}(a-1, a; b-1, b) = 1$ for all markers $a$ and all markers $b \in \mathcal{L}_a$. Hence, $\mathcal{C}$ is $\mathcal{L}$-closed with probability at least $1 - \zeta n^2/2$. Analogously, from the `K-Order` conditions it follows that for any marker $a$ and any marker $b \in \mathcal{R}_b$, $\mathcal{C}(a, a+1; b, b+1) = 1$ with probability at least $1 - \zeta$. Hence, with probability at least $1 - \zeta n^2/2$, $\mathcal{C}(a, a+1; b, b+1) = 1$ for all markers $a$ and all markers $b \in \mathcal{R}_a$. Thus, $\mathcal{C}$ is $\mathcal{R}$-closed with probability at least $1 - \zeta n^2/2$ as well. To sum up, this means that $\mathcal{C}$ is $\mathcal{L}$-closed and $\mathcal{R}$-closed with probability at least $1 - \zeta n^2$.

Now, assume that the `K-Order` algorithm has connected only adjacent markers in its first $k-1$ steps and that $\mathcal{C}$ is $\mathcal{L}$-closed and $\mathcal{R}$-closed. Let $c_\ell$ be the leftmost chain that is connected in the $k$th step. Let $a$ and $c$ be the left and right endpoints, respectively, of the chain $c_\ell$ (note that in the singleton case $c$ is identical to $a$). Let $d$ be the marker immediately to the right of $c$, and let $e$ be any free marker to the right of $d$; see Figure 3.3. Notice that, since $\mathcal{C}$ is $\mathcal{R}$-closed, no free marker to the right of $d$ belongs to $\mathcal{R}_c$. From the `K-Order` condition it thus follows that $\mathcal{C}(c, d; c, e) = -1$ with probability $\geq 1 - \gamma$. For the same reason, in the case where $a$ and $c$ are distinct, the `K-Order` condition implies that $\mathcal{C}(c, d; a, e) = -1$ with probability $\geq 1 - \gamma$. Since $\mathcal{C}$ is $\mathcal{L}$-closed, $a$ does not belong to $\mathcal{L}_d$, which means that if $a$ and $c$ are distinct, $\mathcal{C}(c, d; a, d) = -1$ with probability $\geq 1 - \gamma$ as well.

Let $e_k$ be the number of endmarkers in the $k$th step. In the case where $a$ and $c$ are distinct, the number of endmarkers not belonging to $c_\ell$ is $e_k - 2$, and the probability that an adjacent marker is connected to $c_\ell$ is $\geq 1 - 2\gamma(e_k - 2)$, given that $c_\ell$ is the leftmost chain in the connection. In the case where $a$ and $c$ are identical, the number of endmarkers not belonging to $c_\ell$ is $e_k - 1$, so the probability that an adjacent marker is connected to $c_\ell$ is $\geq 1 - \gamma(e_k - 1)$, given that $c_\ell$ is the leftmost chain in the connection. Since $1 - 2\gamma(e_k - 2) \leq 1 - \gamma(e_k - 1)$ for $e_k \geq 3$, we thus conclude that the probability that an adjacent marker is connected to $c_\ell$ is $\geq 1 - 2\gamma(e_k - 2)$, given that $c_\ell$ is the leftmost chain in the connection.

Let $A_k$ be the event that two adjacent markers are connected in the $k$th step, and let $L(i)$ be the event that the $i$th chain from the left is chosen as the leftmost chain in the connection. Then

$$\begin{aligned}\Pr\big[A_k \mid A_1, \dots, A_{k-1}\big] \\ &= \sum_{i=1}^{n-k+1} \Pr\big[A_k \mid L(i), A_1, \dots, A_{k-1}\big] \cdot \Pr\big[L(i) \mid A_1, \dots, A_{k-1}\big] \\ &\geq (1 - 2\gamma(e_k - 2)) \sum_{i=1}^{n-k+1} \Pr\big[L(i) \mid A_1, \dots, A_{k-1}\big] \\ &= 1 - 2\gamma(e_k - 2).\end{aligned} \tag{3.16}$$

We know that the number of endpoints in step $k$ is at most twice the number of chains in step $k$, i.e., at most $2(n-k+1)$. If we sum up the error probabilities for the $k-1$ steps of the algorithm, we get an upper bound on the error probability $\epsilon$ for the `K-Order` algorithm, given that $\mathcal{C}$ is $\mathcal{L}$-closed and $\mathcal{R}$-closed:

$$\epsilon \leq \sum_{k=1}^{n-1} 2\gamma(e_k - 2) \leq \sum_{k=1}^{n-1} 4\gamma(n-k) = 2\gamma n(n-1) \leq 2\gamma n^2. \tag{3.17}$$

Let $A$ be the event that the `K-Order` algorithm gives a correct order of the markers. What we have shown is that

$$\Pr\big[\mathcal{C} \text{ is } \mathcal{L}\text{-closed} \wedge \mathcal{C} \text{ is } \mathcal{R}\text{-closed}\big] > 1 - \zeta n^2 \tag{3.18}$$

and that

$$\Pr\big[A \mid \mathcal{C} \text{ is } \mathcal{L}\text{-closed} \wedge \mathcal{C} \text{ is } \mathcal{R}\text{-closed}\big] > 1 - 2\gamma n^2. \tag{3.19}$$

Hence the probability that `K-Order` gives the correct order of the markers is

$$\begin{aligned}\Pr\big[A\big] &\geq \Pr\big[A \wedge (\mathcal{C} \text{ is } \mathcal{L}\text{-closed} \wedge \mathcal{C} \text{ is } \mathcal{R}\text{-closed})\big] \\ &> (1 - 2\gamma n^2)(1 - \zeta n^2) \\ &> 1 - 2\gamma n^2 - \zeta n^2. \qquad \square\end{aligned} \tag{3.20}$$

The number of experiments required to make the distance comparator $\widehat{\mathcal{C}}$ satisfy the `K-Order` conditions is naturally dependent on how the markers are arranged on the particular chromosome. Using the distances defined in Definition 3.6, we find expressions for the parameters $\gamma$ and $\zeta$ such that the distance comparator $\widehat{\mathcal{C}}$ satisfies the `K-Order` conditions.

Definition 3.9. *Define the functions* $f_{\mathrm{k}\gamma}(\lambda)$ *and* $f_{\mathrm{k}\zeta}(\lambda)$ *as*

$$f_{\mathrm{k}\gamma}(\lambda) = \min\left\{ \min_{2 \leq i \leq n-1} \{f_{\sigma^L_{i+1}, \ell_{i,i+1}}(\lambda)\}, \min_{1 \leq i \leq n-2} \{f_{\sigma^R_i, \ell_{i,i+1}}(\lambda)\} \right\}, \tag{3.21}$$

$$f_{\mathrm{k}\zeta}(\lambda) = \min_{\substack{1 \leq i \leq n-1 \\ \mathcal{L}_{i+1} \cup \mathcal{R}_i \neq \emptyset}} \{f_{\ell_{i,i+1} - m_i, m_i}(\lambda)\}, \tag{3.22}$$

*and let* $f_{\mathrm{k}}(\lambda) = \min\{f_{\mathrm{k}\gamma}(\lambda), f_{\mathrm{k}\zeta}(\lambda)\}$.

Lemma 3.10. *Given $m$ experiments, the distance comparator $\widehat{\mathcal{C}}$, defined in Definition 2.2, satisfies the `K-Order` condition with parameters*

$$\gamma = e^{-2mp^2q^2(1-(\alpha+\beta))^4 f_{\mathtt{k}\gamma}(\lambda)^2} \quad \text{and} \quad \zeta = e^{-2mp^2q^2(1-(\alpha+\beta))^4 f_{\mathtt{k}\zeta}(\lambda)^2}. \tag{3.23}$$

*Proof.* Lemma 2.3 states that for any four markers $a$, $b$, $c$, and $d$ such that $d(a,d) > d(b,c)$

$$\Pr\bigl[\widehat{\mathcal{C}}(a,d;b,c) \neq 1\bigr] \le e^{-2mp^2q^2(1-(\alpha+\beta))^4 f_{\ell_{ad}-\ell_{bc},\ell_{bc}}(\lambda)^2}. \tag{3.24}$$

Let $a$, $b$, $c$, and $d$ be four markers appearing in this order on the chromosome, and assume that $b$ and $c$ are consecutive, that $d(a,d) > d(b,c)$, and that $d \notin \mathcal{R}_b$. If we let $e = R_b$, it follows from Lemma 2.4 and Definition 3.6 that

$$\begin{aligned}\Pr\bigl[\widehat{\mathcal{C}}(a,d;b,c) \neq 1\bigr] &\le e^{-2mp^2q^2(1-(\alpha+\beta))^4 f_{\ell_{ad}-\ell_{bc},\ell_{bc}}(\lambda)^2}\\ &\le e^{-2mp^2q^2(1-(\alpha+\beta))^4 f_{\ell_{be}-\ell_{bc},\ell_{bc}}(\lambda)^2}\\ &= e^{-2mp^2q^2(1-(\alpha+\beta))^4 f_{\sigma_b^R,\ell_{bc}}(\lambda)^2}.\end{aligned} \tag{3.25}$$

In a similar way we can show that if $a \notin \mathcal{L}_c$, then

$$\Pr\bigl[\widehat{\mathcal{C}}(a,d;b,c) \neq 1\bigr] \le e^{-2mp^2q^2(1-(\alpha+\beta))^4 f_{\sigma_c^L,\ell_{bc}}(\lambda)^2}. \tag{3.26}$$

Hence, from Definition 3.9 it follows that, for any four markers $a$, $b$, $c$, and $d$ appearing in this order on the chromosome such that $b$ and $c$ are consecutive, $d(a,d) > d(b,c)$, and either $d \notin \mathcal{R}_b$ or $a \notin \mathcal{L}_c$,

$$\Pr\bigl[\widehat{\mathcal{C}}(a,d;b,c) = 1\bigr] > 1 - e^{-2mp^2q^2(1-(\alpha+\beta))^4 f_{\mathtt{k}\gamma}(\lambda)^2}. \tag{3.27}$$

Let $a \in [2,n]$ be an arbitrary marker, and assume that $b \in \mathcal{L}_a$. From Lemma 2.4, Definition 3.6, and Definition 3.9 it follows that

$$\begin{aligned}\Pr\bigl[\widehat{\mathcal{C}}(a-1,a;b-1,b) \neq 1\bigr] &\le e^{-2mp^2q^2(1-(\alpha+\beta))^4 f_{\ell_{a-1,a}-\ell_{b-1,b},\ell_{b-1,b}}(\lambda)^2}\\ &\le e^{-2mp^2q^2(1-(\alpha+\beta))^4 f_{\ell_{a-1,a}-m_{a-1},m_{a-1}}(\lambda)^2}\\ &\le e^{-2mp^2q^2(1-(\alpha+\beta))^4 f_{\mathtt{k}\zeta}(\lambda)^2}.\end{aligned} \tag{3.28}$$

In the same way it is possible to show that for any $a \in [1, n-1]$ and $b$ in $\mathcal{R}_a$

$$\begin{aligned}\Pr\bigl[\widehat{\mathcal{C}}(a,a+1;b,b+1) \neq 1\bigr] &\le e^{-2mp^2q^2(1-(\alpha+\beta))^4 f_{\ell_{a,a+1}-m_a,m_a}(\lambda)^2}\\ &\le e^{-2mp^2q^2(1-(\alpha+\beta))^4 f_{\mathtt{k}\zeta}(\lambda)^2}. \qquad \square\end{aligned} \tag{3.29}$$

If we use Lemma 3.10 in combination with Theorem 3.8, we get the following theorem, which gives an upper bound on the number of experiments that is sufficient for the `K-Order` algorithm to give the correct order of the markers.

Theorem 3.11. *Given*

$$m \ge \ln\left(\frac{4n^2}{\epsilon}\right) \frac{1}{2p^2q^2(1-(\alpha+\beta))^4 f_{\mathtt{k}}(\lambda)^2} \tag{3.30}$$

*experiments, the `K-Order` algorithm gives the correct order of the markers with probability greater than $1-\epsilon$.*

*Proof.* From Lemma 3.10 it follows that with

$$m \geq \ln\left(\frac{4n^2}{\epsilon}\right)\frac{1}{2p^2q^2(1-(\alpha+\beta))^4 f_{\mathtt{k}}(\lambda)^2} \tag{3.31}$$

experiments, $\widehat{\mathcal{C}}$ satisfies the `K-Order` condition with parameters $\gamma \leq \epsilon/4n^2$ and $\zeta \leq \epsilon/4n^2$. By Theorem 3.8, this implies that the `K-Order` algorithm will give the correct order of the markers with probability greater than $1-\epsilon$. $\square$

**4. New multiple intensity algorithms.** It is clear from the bounds in Theorems 3.5 and 3.11 that the number of experiments needed for the `P-Order` and `K-Order` algorithms to compute the correct marker order is heavily dependent on the radiation intensity used, or more precisely, on the relation between intensity and the distances compared given by the function $f_{\ell_1,\ell_2}(\lambda)$. Here we present the `Modified-P-Order` and `Modified-K-Order` algorithms which are able to use RH data from several series of experiments, performed with different intensities of the radiation. We show that if the output from $O(\log(n))$ series of experiments with suitably chosen intensities is used, the dependency on the intensity of the radiation can be removed from the performance bounds if we assume that the markers are uniformly distributed.

In each step of both the `K-Order` and `P-Order` algorithms, we look for the minimum distance in a set of given distances (between pairs of markers $\ell_i$). This is done through a sequence of pairwise comparisons between separation probabilities. In the `Modified-P-Order` and `Modified-K-Order` algorithms, each such comparison is made with the estimates calculated from the series of experiments giving the greatest difference in the estimated separation probabilities for the two pairs of markers. We thus get the following distance comparator.

DEFINITION 4.1. *Let $\boldsymbol{\lambda} = (\lambda_1, \ldots, \lambda_t)$ be the intensities used in the $t$ experiment series, and let $\hat{d}_{\lambda_i}(x,y)$ be the distance estimate obtained by applying the distance estimate defined by* (2.5) *to the data from series $i$. For any four markers $a$, $b$, $c$, and $d$, let $\tilde{\lambda}$ be the intensity maximizing the difference in separation probability between $a$, $b$ and $c$, $d$ among the $t$ intensities $\lambda_i$. Define*

$$\mathcal{C}_{t,\boldsymbol{\lambda}}(a,b;c,d) = \begin{cases} 1 & \text{if } \hat{d}_{\tilde{\lambda}}(a,b) > \hat{d}_{\tilde{\lambda}}(c,d), \\ 0 & \text{if } \hat{d}_{\tilde{\lambda}}(a,b) = \hat{d}_{\tilde{\lambda}}(c,d), \\ -1 & \text{if } \hat{d}_{\tilde{\lambda}}(a,b) < \hat{d}_{\tilde{\lambda}}(c,d). \end{cases} \tag{4.1}$$

The use of this distance comparator is motivated by Lemma 2.3 and (2.1). Lemma 2.3 states that if $a$, $b$, $c$, and $d$ are four markers such that $d(a,b) > d(c,d)$, then $\Pr\big[\hat{d}(a,b) \leq \hat{d}(c,d)\big] \leq e^{-O(f_{\ell_{ab}-\ell_{cd},\ell_{cd}}(\lambda)^2)}$. This means that we get the best bound for the error probability when using the intensity $\lambda$ that maximizes $f_{\ell_{ab}-\ell_{cd},\ell_{cd}}(\lambda)$. Unfortunately, this value is unknown, since the distances between the markers are unknown. However, (2.9) implies that the difference in separation probability between $a$, $b$ and $c$, $d$ is proportional to $f_{\ell_{ab}-\ell_{cd},\ell_{cd}}(\lambda)$; we would thus expect the optimal intensity to be the one maximizing this difference.

To analyze the modified algorithms, we will start by showing an upper bound on the probability of misjudging a pair of distances when using the distance comparator $\mathcal{C}_{t,\boldsymbol{\lambda}}$. This bound will be similar to the bound in Lemma 2.3.

Assume that two series of RH experiments have been made, each consisting of $m$ experiments. In the first series the intensity $\lambda_1$ was used, and in the second series the intensity $\lambda_2$ was used. For $i = 1, \ldots, m$ and $j = 1, 2$ we define $X_i^j = 1$ if $a, b$ but not

$c, d$ are separated in the $i$th experiment of series $j$; $X_i^j = -1$ if $c, d$ but not $a, b$ are separated in the $i$th experiment of series $j$; and $X_i^j = 0$ otherwise. Furthermore, for $j = 1, 2$, we let

$$Y_j = \sum_{i=1}^{m} X_i^j. \tag{4.2}$$

Equation (2.10), states that in this case

$$E[Y_j] = 2mpq(1-(\alpha+\beta))^2 f_{\ell_{ab}-\ell_{cd},\ell_{cd}}(\lambda_j). \tag{4.3}$$

Assume that $d(a, b) > d(c, d)$, and that $Y_1 > 0$, i.e., that the series of experiments using intensity $\lambda_1$ has given the correct ordering of the distances. We will bound the probability that, in the algorithm, the results from the second series of experiments will mislead us into drawing the incorrect conclusion about the two distances $d(a, b)$ and $d(c, d)$. This will happen when $Y_2 < -Y_1$, i.e., when $Y_1 + Y_2 < 0$. Let

$$Z = \sum_{i=1}^{m} X_i^1 + \sum_{i=1}^{m} X_i^2 = Y_1 + Y_2. \tag{4.4}$$

Since $d(a, b) > d(c, d)$, it is clear that

$$E[Z] = E[Y_1] + E[Y_2] > 0. \tag{4.5}$$

Using the Hoeffding inequality and (4.3), we see that

$$\begin{aligned} \Pr\big[Z < 0\big] &= \Pr\big[E[Z] - Z \geq E[Z]\big] = \Pr\left[\frac{E[Z]-Z}{2m} \geq \frac{E[Z]}{2m}\right] \\ &\leq e^{-2E[Z]^2/8m} = e^{-2(E[Y_1]+E[Y_2])^2/8m} \\ &= e^{-mp^2q^2(1-(\alpha+\beta))^4(f_{\ell_{ab}-\ell_{cd},\ell_{cd}}(\lambda_1)+f_{\ell_{ab}-\ell_{cd},\ell_{cd}}(\lambda_2))^2}. \end{aligned} \tag{4.6}$$

This inequality will be used to prove the following lemma.

LEMMA 4.2. *Let $a, b, c$ and $d$ be four markers such that $d(a, b) > d(c, d)$. Then*

$$\Pr\big[\mathcal{C}_{\boldsymbol{\lambda},t}(a, b; c, d) \neq 1\big] \leq te^{-mp^2q^2(1-(\alpha+\beta))^4 f_{\ell_{ab}-\ell_{cd},\ell_{cd}}(\lambda^*)^2}, \tag{4.7}$$

*where $\lambda^*$ is the intensity maximizing $f_{\ell_{ab}-\ell_{cd},\ell_{cd}}(\lambda)$ among the $t$ intensities $\lambda_1, \dots, \lambda_t$ in $\boldsymbol{\lambda}$.*

*Proof.* Let $B_1$ be the event that the decision based on the series of experiments using intensity $\lambda^*$ is incorrect, and let $B_2$ be the event that the series of experiments using intensity $\lambda^*$ is correct but some other series is chosen and the decision based on that series is incorrect. From Lemma 2.3 follows that

$$\Pr\big[B_1\big] \leq e^{-2mp^2q^2(1-(\alpha+\beta))^4 f_{\ell_{ab}-\ell_{cd},\ell_{cd}}(\lambda^*)^2}, \tag{4.8}$$

and from (4.6) follows that

$$\Pr\big[B_2\big] \leq \sum_{\lambda_i \neq \lambda^*} e^{-mp^2q^2(1-(\alpha+\beta))^4(f_{\ell_{ab}-\ell_{cd},\ell_{cd}}(\lambda^*)+f_{\ell_{ab}-\ell_{cd},\ell_{cd}}(\lambda_i))^2}. \tag{4.9}$$

This means that

$$\begin{aligned}\Pr\big[\mathcal{C}_{\boldsymbol{\lambda},t}(a,b;c,d)\neq 1\big] &\le \Pr\big[B_1\big]+\Pr\big[B_1\big]\\ &\le e^{-2mp^2q^2(1-(\alpha+\beta))^4 f_{\ell_{ab}-\ell_{cd},\ell_{cd}}(\lambda^*)^2}\\ &\qquad+\sum_{\lambda_i\neq\lambda^*} e^{-mp^2q^2(1-(\alpha+\beta))^4(f_{\ell_{ab}-\ell_{cd},\ell_{cd}}(\lambda^*)+f_{\ell_{ab}-\ell_{cd},\ell_{cd}}(\lambda_i))^2}\\ &\le te^{-mp^2q^2(1-(\alpha+\beta))^4 f_{\ell_{ab}-\ell_{cd},\ell_{cd}}(\lambda^*)^2}. \qquad \square\end{aligned} \tag{4.10}$$

To eliminate the dependency on the intensity $\lambda$ from the bound in Lemma 4.2 we need to investigate the function $f_{\ell_1,\ell_2}(\lambda)$ in more detail. Differentiation of the function $f_{\ell_1,\ell_2}(\lambda)$ with respect to $\lambda$ yields that

$$\frac{df_{\ell_1,\ell_2}(\lambda)}{d\lambda}=(\ell_1+\ell_2)e^{-\lambda(\ell_1+\ell_2)}-\ell_2e^{-\lambda\ell_2}, \tag{4.11}$$

$$\frac{d^2f_{\ell_1,\ell_2}(\lambda)}{d\lambda^2}=-(\ell_1+\ell_2)^2e^{-\lambda(\ell_1+\ell_2)}+\ell_2^2e^{-\lambda\ell_2}. \tag{4.12}$$

We observe that

$$\hat\lambda=\frac{1}{\ell_1}\ln\frac{\ell_1+\ell_2}{\ell_2} \tag{4.13}$$

is the only stationary point for $f_{\ell_1,\ell_2}(\lambda)$ when $\ell_1,\ell_2>0$, and that this stationary point is in fact a maximum since

$$\frac{d^2f_{\ell_1,\ell_2}(\hat\lambda)}{d\lambda^2}=-\ell_1\ell_2\left(\frac{\ell_2}{\ell_1+\ell_2}\right)^{\frac{\ell_2}{\ell_1}}<0 \tag{4.14}$$

for $\ell_1,\ell_2>0$. Furthermore, we see that the corresponding optimal value is

$$f_{\ell_1,\ell_2}(\hat\lambda)=\left(\frac{\ell_1}{\ell_1+\ell_2}\right)\left(\frac{\ell_2}{\ell_1+\ell_2}\right)^{\frac{\ell_2}{\ell_1}}. \tag{4.15}$$

If an intensity $\tilde\lambda$ is sufficiently close to the optimal intensity $\hat\lambda$, the value of $f_{\ell_1,\ell_2}(\tilde\lambda)$ should not be too far from the optimal value $f_{\ell_1,\ell_2}(\hat\lambda)$. The following lemma states that for $\tilde\lambda\in[\hat\lambda/2,2\hat\lambda]$ a lower bound on $f_{\ell_1,\ell_2}(\tilde\lambda)$ can be given that is independent of the intensity.

LEMMA 4.3. *Let $\hat\lambda$ be the unique optimum for $f_{\ell_1,\ell_2}(\lambda)$, i.e.,*

$$\hat\lambda=\frac{1}{\ell_1}\ln\frac{\ell_1+\ell_2}{\ell_2}. \tag{4.16}$$

*Then, for any $\tilde\lambda\in[\hat\lambda/2,2\hat\lambda]$,*

$$f_{\ell_1,\ell_2}(\tilde\lambda)\ge\frac{1}{2e^2}\frac{\ell_1}{\ell_1+\ell_2}. \tag{4.17}$$

*Proof.* Since $f_{\ell_1,\ell_2}(\lambda)$ has a single stationary point which is a local maximum, we know that for any $\tilde\lambda$ such that $\tilde\lambda\in[\hat\lambda/2,2\hat\lambda]$,

$$f_{\ell_1,\ell_2}(\tilde\lambda)\ge\min\left\{f_{\ell_1,\ell_2}\left(\frac{\hat\lambda}{2}\right),f_{\ell_1,\ell_2}(2\hat\lambda)\right\}. \tag{4.18}$$

From the definition of $f_{\ell_1,\ell_2}(\lambda)$ it follows that for any $k > 0$

$$f_{\ell_1,\ell_2}(k\hat{\lambda}) = \left(1 + \frac{\ell_1}{\ell_2}\right)^{-k\frac{\ell_2}{\ell_1}} \left(1 - \left(1 - \frac{\ell_1}{\ell_1 + \ell_2}\right)^k\right). \tag{4.19}$$

Since we know that $(1 + 1/x)^{-kx} \geq e^{-k}$ for $x > 0$, this means that

$$f_{\ell_1,\ell_2}(k\hat{\lambda}) \geq e^{-k} \left(1 - \left(1 - \frac{\ell_1}{\ell_1 + \ell_2}\right)^k\right). \tag{4.20}$$

If we insert $k = 1/2$ and $k = 2$ into this equation and use the inequality

$$\min\{1 - (1 - x)^2, 1 - (1 - x)^{1/2}\} \geq \frac{x}{2} \tag{4.21}$$

for $0 < x < 1$, we thus get that

$$f_{\ell_1,\ell_2}(\tilde{\lambda}) \geq \frac{1}{2e^2} \frac{\ell_1}{\ell_1 + \ell_2}. \qquad \square \tag{4.22}$$

Lemma 4.3 implies that by using a range of intensities, the choice of intensities is eliminated completely from the bound in Lemma 4.2.

LEMMA 4.4. *Let a, b, c, and d be four markers such that* $d(a, b) > d(c, d)$. *Let* $\boldsymbol{\lambda} = (\lambda_1, \ldots , \lambda_t)$, *where* $\lambda_i = 4\lambda_{i-1}$, *for* $2 \leq i \leq t$. *Let* $\hat{\lambda}$ *be the intensity maximizing* $f_{\ell_{ab}-\ell_{cd},\ell_{cd}}(\lambda)$, *and assume that* $\lambda_1 \leq \hat{\lambda} \leq \lambda_t$. *Then,*

$$\Pr\left[\mathcal{C}_{\boldsymbol{\lambda},t}(a, b; c, d) \neq 1\right] \leq te^{-mp^2q^2(1-(\alpha+\beta))^4\left(\frac{\ell_{ab}-\ell_{cd}}{2e^2\ell_{ab}}\right)^2}. \tag{4.23}$$

*Proof.* By assumption, the intensities used in the $t$ series will satisfy $\lambda_{i+1} = 4\lambda_i$, for $i = 1, \ldots , t - 1$. Since the optimal intensity $\hat{\lambda}$ satisfies $\lambda_1 \leq \hat{\lambda} \leq \lambda_t$, this means that there is some intensity $\tilde{\lambda} \in \{\lambda_i\}$ such that $\hat{\lambda}/2 \leq \tilde{\lambda} \leq 2\hat{\lambda}$. Let $\lambda^*$ be the intensity maximizing $f_{\ell_{ab}-\ell_{cd},\ell_{cd}}(\lambda)$ among the $t$ intensities $\lambda_1, \ldots , \lambda_t$. Then, from Lemma 4.3 it follows that

$$f_{\ell_{ab}-\ell_{cd},\ell_{cd}}(\lambda^*) \geq f_{\ell_{ab}-\ell_{cd},\ell_{cd}}(\tilde{\lambda}) \geq \frac{\ell_{ab} - \ell_{cd}}{2e^2\ell_{ab}}. \tag{4.24}$$

If we use this inequality in Lemma 4.2, we get that

$$\begin{aligned} \Pr\left[\mathcal{C}_{\boldsymbol{\lambda},t}(a, b; c, d) \neq 1\right] &\leq te^{-mp^2q^2(1-(\alpha+\beta))^4 f_{\ell_{ab}-\ell_{cd},\ell_{cd}}(\lambda^*)^2} \\ &\leq te^{-mp^2q^2(1-(\alpha+\beta))^4\left(\frac{\ell_{ab}-\ell_{cd}}{2e^2\ell_{ab}}\right)^2}. \qquad \square \end{aligned} \tag{4.25}$$

To be able to use Lemma 4.4 in the same way as we used Lemma 2.3 in the proof of the original `P-Order` and `K-Order` algorithms, we need to make sure that the optimal intensity $\hat{\lambda}$ satisfies $\lambda_1 \leq \hat{\lambda} \leq \lambda_t$, for all parameters $\ell_1$ and $\ell_2$ of the function $f_{\ell_1,\ell_2}(\lambda)$ considered in the analyses. The following lemma shows that the constraint on the optimal intensity can be transformed into a constraint on the two parameters $\ell_1$ and $\ell_2$.

LEMMA 4.5. *Let* $\ell_1$ *and* $\ell_2$ *be two distances such that* $0 < \ell_{\min} \leq \ell_1, \ell_2 \leq \ell_{\max}$, *and let* $\hat{\lambda}$ *be the optimal value of* $f_{\ell_1,\ell_2}(\lambda)$. *Then* $\hat{\lambda} \in [\ln 2/\ell_{\max}, \ln 2/\ell_{\min}]$.

*Proof.* Above, we observed that $f_{\ell_1,\ell_2}(\lambda)$ is maximized by $\hat{\lambda} = \ln((\ell_1+\ell_2)/\ell_2)/\ell_1$. Differentiation yields that

$$\frac{d\hat{\lambda}}{d\ell_1} = \frac{1}{\ell_1^2}\left(\frac{1}{1+\ell_2/\ell_1} - \ln\left(1+\frac{\ell_1}{\ell_2}\right)\right), \tag{4.26}$$

$$\frac{d\hat{\lambda}}{d\ell_2} = -\frac{1}{\ell_2+\ell_1\ell_2}. \tag{4.27}$$

It follows immediately from (4.27) that $\hat{\lambda}$ is decreasing when $\ell_2$ is increasing. To see that $\hat{\lambda}$ is also decreasing when $\ell_1$ is increasing, we observe that

$$\frac{d}{dx}\left(\frac{1}{1+1/x} - \ln(1+x)\right) = \frac{1}{(1+x)^2} - \frac{1}{1+x} < 0, \tag{4.28}$$

for $x > 0$, and that

$$\lim_{x\to 0}\left(\frac{1}{1+1/x} - \ln(1+x)\right) = 0. \tag{4.29}$$

Hence, if $0 < \ell_{\min} \le \ell_1, \ell_2 \le \ell_{\max}$, we are guaranteed that $\ln 2/\ell_{\max} \le \hat{\lambda} \le \ln 2/\ell_{\min}$. $\square$

If we look carefully at the analyses of the `P-Order` and `K-Order` algorithms, we observe that the parameters $\ell_1$ and $\ell_2$ of the function $f_{\ell_1,\ell_2}(\lambda)$ always are less than or equal to the longest distance between any pair of markers, and greater than or equal to the shortest distance between any pair of markers. The longest distance between any pair of markers can obviously be bounded by the length of the genome. Furthermore, if we assume that the markers are uniformly distributed along the genome, we can derive a probabilistic lower bound on the shortest distance between any two markers.

LEMMA 4.6. *Let $L$ be the length of a genome containing $n$ uniformly distributed markers. Then the distance between any two markers is at least $L/(2n^{2+\delta})$, with probability* $1 - n^{-\delta}$.

*Proof.* Divide the genome into $2n^{2+\delta}$ disjoint intervals, each of length $L/(2n^{2+\delta})$. Define the $\binom{n}{2}$ random variables $X_{ij}$ in the following way:

$$X_{ij} = \begin{cases} 1 & \text{if } i \text{ and } j \text{ are in the same or neighboring subintervals,} \\ 0 & \text{otherwise.} \end{cases} \tag{4.30}$$

Since the markers are assumed to be uniformly distributed, we know that

$$\Pr\big[X_{ij} = 1\big] \le \frac{3}{2n^{2+\delta}}. \tag{4.31}$$

Let

$$Y = \sum_{i<j} X_{ij}. \tag{4.32}$$

It is clear that $Y \ge 1$ if and only if some pair of markers are in the same or neighboring subintervals. From the Markov inequality it follows that

$$\Pr\big[Y \ge 1\big] \le E[Y] \le \binom{n}{2}\frac{3}{2n^{2+\delta}} < \frac{1}{n^{\delta}}. \tag{4.33}$$

Hence, the distance between any two markers will be at least $L/(2n^{2+\delta})$, with probability $1 - n^{-\delta}$. $\square$

**4.1. Analyzing the `Modified-P-Order` algorithm.** The similarity between the bound in Lemmas 2.3 and 4.4 makes it possible to carry through derivations similar to those in section 3. The role of the intensity-dependent function $f_{\mathtt{P}}(\lambda)$ will be played by the function $f^*_{\mathtt{P}}$ defined below.

DEFINITION 4.7. *Define the parameter $f^*_{\mathtt{P}}$ as*

$$f^*_{\mathtt{P}} = \min_{1<i<n} \left\{ \frac{\ell_{i-1,i}}{\ell_{i-1,i+1}}, \frac{\ell_{i,i+1}}{\ell_{i-1,i+1}} \right\}. \tag{4.34}$$

LEMMA 4.8. *Assume that $n$ markers are uniformly distributed along a chromosome of length $L$. Assume that $t = \lceil (1+\delta/2)\log(n) + 3/2 \rceil$ series of experiments have been made, each consisting of $m$ experiments using intensities $\boldsymbol{\lambda} = (\lambda_1, \dots, \lambda_t)$, where $\lambda_i = 4^{i-1}\ln 2/L$. Then, with probability greater than $1 - n^{-\delta}$, the distance comparator $\mathcal{C}_{\boldsymbol{\lambda},t}$ satisfies the* `P-Order` *condition with parameter*

$$te^{-mp^2q^2(1-(\alpha+\beta))^4\left(\frac{f^*_{\mathtt{P}}}{2e^2}\right)^2}. \tag{4.35}$$

*Proof.* From Lemmas 4.5 and 4.6 follows that, with probability greater than $1-n^{-\delta}$, the optimal intensities $\hat{\lambda}$, for all choices of $\ell_1$ and $\ell_2$ such that $\ell_{\min} \le \ell_1, \ell_2 \le L$, will satisfy $\hat{\lambda} \in [\lambda_1, \lambda_t]$, where $\ell_{\min}$ is the shortest distance between any two markers and $t$ is chosen as above.

Let $a$, $b$, $c$, and $d$ be four markers, appearing in this order on the genome, such that $b$ and $c$ are consecutive and $d(a,d) > d(b,c)$. Without loss of generality, we assume that $d \ne c$. Let $e$ be the marker immediately to the right of $c$. Note that $e$ may be identical to the marker $d$. Then, under the condition above, it follows from Lemma 4.4 that

$$\begin{aligned} \Pr\left[\mathcal{C}_{\boldsymbol{\lambda},t}(a,d;b,c) \ne 1\right] &\le te^{-mp^2q^2(1-(\alpha+\beta))^4\left(\frac{\ell_{ad}-\ell_{bc}}{2e^2\ell_{ad}}\right)^2} \\ &\le te^{-mp^2q^2(1-(\alpha+\beta))^4\left(\frac{\ell_{ce}}{2e^2\ell_{be}}\right)^2} \\ &\le te^{-mp^2q^2(1-(\alpha+\beta))^4\left(\frac{f^*_{\mathtt{P}}}{2e^2}\right)^2}, \end{aligned} \tag{4.36}$$

where the last inequality follows from the fact that the markers $b$, $c$, $e$ are consecutive, together with Definition 4.7. □

THEOREM 4.9. *Assume that $n$ markers are uniformly distributed along a chromosome of length $L$. Assume that $t = \lceil (1+\delta/2)\log(n) + 3/2 \rceil$ series of experiments have been made using intensities $\boldsymbol{\lambda} = (\lambda_1, \dots, \lambda_t)$, where $\lambda_i = 4^{i-1}\ln 2/L$, each consisting of*

$$m \ge \ln\left(\frac{tn^2}{\epsilon}\right) \frac{4e^4}{p^2q^2(1-(\alpha+\beta))^4(f^*_{\mathtt{P}})^2} \tag{4.37}$$

*experiments. Then the* `Modified-P-Order` *algorithm, i.e., the* `P-Order` *algorithm using the distance comparator $\mathcal{C}_{\boldsymbol{\lambda},t}$, gives the correct order of the $n$ markers with probability greater than $1 - \epsilon - n^{-\delta}$.*

*Proof.* From Lemma 4.8 it follows that, given

$$m \ge \ln\left(\frac{tn^2}{\epsilon}\right) \frac{4e^4}{p^2q^2(1-(\alpha+\beta))^4\left(f^*_{\mathtt{P}}\right)^2} \tag{4.38}$$

experiments, the distance comparator $\mathcal{C}_{\boldsymbol{\lambda},t}$ satisfies the `P-Order` condition with parameter $\leq \epsilon/n^2$ with probability greater than $1-n^{-\delta}$. By Theorem 3.2, this implies that the `P-Order` algorithm using this distance comparator will compute the correct order of the markers with probability greater than $1-\epsilon$. Hence, the `Modified-P-Order` algorithm will compute the correct order of the markers with probability greater than $1-\epsilon-n^{-\delta}$. □

**4.2. Analyzing the `Modified-K-Order` algorithm.** The equivalents of the functions $f_{\mathtt{k}\gamma}(\lambda)$, $f_{\mathtt{k}\zeta}(\lambda)$, and $f_{\mathtt{k}}(\lambda)$ in section 3 will in the modified case be denoted $f^*_{\mathtt{k}\gamma}$, $f^*_{\mathtt{k}\zeta}$, and $f^*_{\mathtt{k}}$, respectively.

DEFINITION 4.10. *Define the parameters $f^*_{\mathtt{k}\gamma}$ and $f^*_{\mathtt{k}\zeta}$ as*

$$f^*_{\mathtt{k}\gamma} = \min\left\{\min_{2\leq i\leq n-1}\left\{\frac{\sigma^L_{i+1}}{\sigma^L_{i+1}+\ell_{i,i+1}}\right\}, \min_{1\leq i\leq n-2}\left\{\frac{\sigma^R_i}{\sigma^R_i+\ell_{i,i+1}}\right\}\right\}, \tag{4.39}$$

$$f^*_{\mathtt{k}\zeta} = \min_{\substack{1\leq i\leq n-1\\ \mathcal{L}_{i+1}\cup\mathcal{R}_i\neq\emptyset}}\left\{\frac{\ell_{i,i+1}-m_i}{\ell_{i,i+1}}\right\}, \tag{4.40}$$

*and let $f^*_{\mathtt{k}} = \min\{f_{\mathtt{k}\gamma}, f_{\mathtt{k}\zeta}\}$.*

LEMMA 4.11. *Assume that $n$ markers are uniformly distributed along a chromosome of length $L$. Assume that $t = \lceil(1+\delta/2)\log(n)+3/2\rceil$ series of experiments have been made, each consisting of $m$ experiments, using intensities $\boldsymbol{\lambda} = (\lambda_1,\dots,\lambda_t)$, where $\lambda_i = 4^{i-1}\ln 2/L$. Then, with probability greater than $1-n^{-\delta}$, the distance comparator $\mathcal{C}_{\boldsymbol{\lambda},t}$ satisfies the `K-Order` conditions with parameters*

$$\gamma = te^{-mp^2q^2(1-(\alpha+\beta))^4\left(\frac{f^*_{\mathtt{k}\gamma}}{2e^2}\right)^2} \quad \text{and} \quad \zeta = te^{-mp^2q^2(1-(\alpha+\beta))^4\left(\frac{f^*_{\mathtt{k}\zeta}}{2e^2}\right)^2}. \tag{4.41}$$

*Proof.* From Lemmas 4.5 and 4.6 it follows that, with probability greater than $1-n^{-\delta}$, the optimal intensities $\hat{\lambda}$, for all choices of $\ell_1$ and $\ell_2$ such that $\ell_{\min}\leq \ell_1,\ell_2\leq L$, will satisfy $\hat{\lambda}\in[\lambda_1,\lambda_t]$, where $\ell_{\min}$ is the shortest distance between any two markers and $t$ is chosen as above.

Let $a$, $b$, $c$, and $d$ be four markers appearing in this order on the genome; assume that $b$ and $c$ are consecutive; assume that $d(a,d) > d(b,c)$; and assume that $d\notin\mathcal{R}_b$. If we let $e = R_b$, it follows from Definition 3.6 and Lemma 4.4 that, under the condition above,

$$\begin{aligned}\Pr\left[\mathcal{C}_{\boldsymbol{\lambda},t}(a,d;b,c)\neq 1\right] &\leq te^{-mp^2q^2(1-(\alpha+\beta))^4\left(\frac{\ell_{ad}-\ell_{bc}}{2e^2\ell_{ad}}\right)^2}\\ &\leq te^{-mp^2q^2(1-(\alpha+\beta))^4\left(\frac{\ell_{be}-\ell_{bc}}{2e^2\ell_{be}}\right)^2}\\ &= te^{-mp^2q^2(1-(\alpha+\beta))^4\left(\frac{\sigma^R_b}{2e^2(\sigma^R_b+\ell_{bc})}\right)^2}.\end{aligned} \tag{4.42}$$

In a similar way we can show that if $a\notin\mathcal{L}_c$, then

$$\Pr\left[\mathcal{C}_{\boldsymbol{\lambda},t}(a,d;b,c)\neq 1\right] \leq te^{-mp^2q^2(1-(\alpha+\beta))^4\left(\frac{\sigma^L_c}{2e^2(\sigma^L_c+\ell_{bc})}\right)^2}. \tag{4.43}$$

Hence, from Definition 4.10 it follows that, for any four markers $a$, $b$, $c$, and $d$ appearing in this order on the genome such that $b$ and $c$ are consecutive, $d(a,d) > d(b,c)$, and either $d\notin\mathcal{R}_b$ or $a\notin\mathcal{L}_c$,

$$\Pr\left[\mathcal{C}_{\boldsymbol{\lambda},t}(a,d;b,c) = 1\right] \geq 1 - te^{-mp^2q^2(1-(\alpha+\beta))^4\left(\frac{f^*_{\mathtt{k}\gamma}}{2e^2}\right)^2}. \tag{4.44}$$

Let $a \in [2, n]$ be an arbitrary marker, and assume that $b \in \mathcal{L}_a$. From the definition of $\mathcal{L}_a$ it follows that $\ell_{a-1,a} - \ell_{b-1,b} \geq \ell_{b-1,b}$. Hence, from Definition 3.6, Definition 4.10, and Lemma 4.4 it follows, under the condition above, that

$$\begin{aligned}\Pr\big[\mathcal{C}_{\boldsymbol{\lambda},t}(a-1,a;b-1,b) \neq 1\big] &\leq te^{-mp^2q^2(1-(\alpha+\beta))^4\left(\frac{\ell_{a-1,a}-\ell_{b-1,b}}{2e^2\ell_{a-1,a}}\right)^2} \\ &\leq te^{-mp^2q^2(1-(\alpha+\beta))^4\left(\frac{\ell_{a-1,a}-m_a}{2e^2\ell_{a-1,a}}\right)^2} \\ &\leq te^{-mp^2q^2(1-(\alpha+\beta))^4\left(\frac{f^*_{\mathtt{k}\zeta}}{2e^2}\right)^2}.\end{aligned} \tag{4.45}$$

In the same way it is possible to show that for any $a \in [1, n-1]$ and $b$ in $\mathcal{R}_a$

$$\Pr\big[\mathcal{C}_{\boldsymbol{\lambda},t}(a,a+1;b,b+1) \neq 1\big] \leq te^{-mp^2q^2(1-(\alpha+\beta))^4\left(\frac{f^*_{\mathtt{k}\zeta}}{2e^2}\right)^2}. \qquad \square \tag{4.46}$$

THEOREM 4.12. *Assume that $n$ markers are uniformly distributed along a chromosome of length $L$. Assume that $t = \lceil(1 + \delta/2)\log(n) + 3/2\rceil$ series of experiments have been made using intensities $\boldsymbol{\lambda} = (\lambda_1, \ldots, \lambda_t)$, where $\lambda_i = 4^{i-1}\ln 2/L$, each consisting of*

$$m \geq \ln\left(\frac{4tn^2}{\epsilon}\right)\frac{4e^4}{p^2q^2(1-(\alpha+\beta))^4(f^*_{\mathtt{k}})^2} \tag{4.47}$$

*experiments. Then the* `Modified-K-Order` *algorithm, i.e., the* `K-Order` *algorithm using the distance comparator $\mathcal{C}_{\boldsymbol{\lambda},t}$, gives the correct order of the $n$ markers with probability greater than $1 - \epsilon - n^{-\delta}$.*

*Proof.* From Lemma 4.11 it follows that, if the number of experiments $m$ satisfies (4.47), then the distance comparator $\mathcal{C}_{\boldsymbol{\lambda},t}$ satisfies the `K-Order` conditions with parameters $\gamma \leq \epsilon/4n^2$ and $\zeta \leq \epsilon/4n^2$ with probability greater than $1 - n^{-\delta}$. By Theorem 3.8, this implies that the `K-Order` algorithm using this distance comparator will compute the correct order of the markers with probability greater than $1 - \epsilon$. Hence, the `Modified-K-Order` algorithm will compute the correct order of the markers with probability greater than $1 - \epsilon - n^{-\delta}$. $\square$

**5. Discussion.** In [2], Ben-Dor and Chor presented the `P-Order` and `K-Order` algorithms for the RH problem. As support for the algorithms, they also proved an upper bound on the number of RH experiments sufficient for both algorithms to compute the correct marker order with high probability.

We have improved the analysis of the `P-Order` and `K-Order` algorithms. By taking into account what decisions the two algorithms actually need to make, we derive tighter bounds than those in [2]. The major improvement is the replacement of $f_{\delta_{\min},\delta_{\max}}(\lambda)$ with $f_{\mathtt{P}}(\lambda)$ and $f_{\mathtt{k}}(\lambda)$, respectively. It is easy to see from their definitions that $f_{\delta_{\min},\delta_{\max}}(\lambda) \leq f_{\mathtt{P}}(\lambda) \leq f_{\mathtt{k}}(\lambda)$. In fact, for any constant $c > 0$ it is possible to construct marker arrangements such that $f_{\mathtt{k}}(\lambda)/f_{\mathtt{P}}(\lambda) > c$ and $f_{\mathtt{P}}(\lambda)/f_{\delta_{\min},\delta_{\max}}(\lambda) > c$. The new analysis thus suggests that the `K-Order` algorithm is to be preferred to the `P-Order` algorithm, a suggestion which is confirmed by experimental results not shown here. The introduction of distance comparators has allowed us to formulate very general results. The bounds in Theorems 3.2 and 3.8 apply to any `P-Order`- and `K-Order`-like strategy for ordering points on the line, such that the probability for errors in the distance comparisons can be bounded.

The new analysis of the `P-Order` and `K-Order` algorithms shows that the intensity of the radiation used in the RH experiments is less suitable for some distance comparisons than for others. This suggests that the use of RH data produced with several different intensities would improve the stability of the algorithms with respect to the choice of intensity. The `Modified-P-Order` and `Modified-K-Order` algorithms are modified versions of the original `P-Order` and `K-Order` algorithms, designed to be able to use multiple intensity data. We show that using RH data from $O(\log n)$ series of experiments, produced with suitable intensities, the dependency on the choice of intensity can be removed from the performance bounds, under the assumption that the markers are uniformly distributed.

Although the analysis of the modified algorithms assumes that a specified range of intensities is used, the algorithms can be used on any experimental data involving different intensities. Experimental studies on synthetic data (omitted here) suggest that the `Modified-K-Order` algorithm works especially well in practice. The use of only three different intensities makes the algorithm more stable against the effects of the choice of intensity. However, the most important conclusion that can be drawn from these results is that the use of multiple intensity data does have a positive effect on the performance of `P-Order`- and `K-Order`-like RH algorithms.

**6. Acknowledgment.** We would like to thank Johan Håstad for useful ideas and fruitful discussions.

## REFERENCES

[1] R. Agarwala, D. L. Applegate, D. Maglott, G. D. Schuler, and A. A. Schäffer, *A fast and scalable radiation hybrid map construction and integration strategy*, Genome Res., 10 (2000), pp. 350–364.

[2] A. Ben-Dor and B. Chor, *On constructing radiation hybrid maps*, J. Comput. Biol., 4 (1997), pp. 517–533.

[3] A. Ben-Dor, B. Chor, and D. Pelleg, *RHO—Radiation hybrid ordering*, Genome Res., 10 (2000), pp. 365–378.

[4] D. T. Bishop and G. P. Crockford, *Comparison of radiation hybrid mapping and linkage mapping*, Cytogeret. Cell Genet., 59 (1992), pp. 93–95.

[5] T. H. Cormen, C. E. Leiserqson, and R. L. Rivest, *Introduction to Algorithms*, MIT Press, Cambridge, MA, McGraw-Hill, New York, 1989.

[6] R. Guyon, T. D. Lorentzen, C. Hitte, L. Kim, E. Cadieu, H. G. Parker, P. Quignon, J. K. Lowe, C. Renier, B. Gelfenbeyn, F. Vignaux, H. B. DeFrance, S. Gloux, G. G. Mahairas, C. Andre, F. Galibert, and E. A. Ostrander, *A* 1*-Mb resolution radiation hybrid map of the canine genome*, Proc. Natl. Acad. Sci. USA, 100 (2003), pp. 5296–5301.

[7] J. Håstad, L. Ivansson, and J. Lagergren, *Fitting points on the real line and its application to RH mapping*, Lecture Notes in Comput. Sci. 1461, Springer-Verlag, New York, 1998, pp. 465–476.

[8] C. Hitte, T. D. Lorentzen, R. Guyon, L. Kim, E. Cadieu, H. G. Parker, P. Quignon, J. K. Lowe, B. Gelfenbeyn, C. Andre, E. A. Ostrander, and F. Galibert, *Comparison of MultiMap and TSP/CONCORDE for constructing radiation hybrid maps*, J. Heredity, 94 (2003), pp. 9–13.

[9] W. Hoeffding, *Probability inequalities for sums of bounded random variables*, J. Amer. Statist. Assoc., 58 (1963), pp. 13–30.

[10] K. Lange, M. Boehnke, D. R. Cox, and K. L. Lunetta, *Statistical methods for polyploid radiation hybrid mapping*, Genome Res., 5 (1995), pp. 136–150.

[11] K. L. Lunetta, M. Boehnke, K. Lange, and D. R. Cox, *Selected locus and multiple panel models for radiation hybrid mapping*, Am. J. Hum. Genet., 59 (1996), pp. 717–725.

[12] M. Menotti-Raymond, V. A. David, R. Agarwala, A. A. Schäffer, R. Stephens, S. J. O'Brien, and W. J. Murphy, *Radiation hybrid mapping of* 304 *novel microsatellites in the domestic cat genome*, Cytogenet. Genome Res., 102 (2003), pp. 272–276.

[13] D. Slonim, L. Kruglyak, L. Stein, and E. Lander, *Building human genome maps with radiation hybrids*, in Proceedings of the First International Conference on Computational Molecular Biology, 1997, ACM Press, New York, pp. 277–286.

SIAM J. COMPUT. © 2004 Society for Industrial and Applied Mathematics
Vol. 34, No. 1, pp. 109–117

# PERFECTNESS IS AN ELUSIVE GRAPH PROPERTY[*]

STEFAN HOUGARDY[†] AND ANNEGRET WAGLER[‡]

**Abstract.** A graph property is called elusive (or evasive) if every algorithm for testing this property has to read in the worst case $\binom{n}{2}$ entries of the adjacency matrix of the given graph. Several graph properties have been shown to be elusive, e.g., planarity or $k$-colorability. A famous conjecture of Karp says that every nontrivial monotone graph property is elusive. We prove that a nonmonotone but hereditary graph property is elusive: perfectness.

**Key words.** perfect graph, elusiveness, evasiveness, graph property testing

**AMS subject classifications.** 68Q17, 68Q25

**DOI.** 10.1137/S0097539703426799

## 1. Introduction.

Given a graph property, consider the following two-player game to define elusiveness. Player A wants to know whether an unknown simple graph on a given node set has the graph property in question by asking Player B one pair of nodes at a time whether this pair of nodes is an edge. At each stage Player A makes full use of the information of edges and nonedges he has up to that point in order to decide whether the graph has the property or not. Player A wants to minimize the number of his questions; Player B wants to force him to ask as many questions as possible. The number of questions needed for the decision if both players play optimally from their point of view is the recognition complexity of the studied graph property. The property is said to be *elusive* (or also *evasive*) if there is a strategy enabling Player B to force Player A to test *every* pair of nodes, respectively, to ask all possible $\binom{n}{2}$ questions before coming to a decision. (More precisely, such a strategy has to exist for all nontrivial cases, i.e., for all $n$ such that there are graphs on $n$ nodes with and without the studied property.)

Several graph properties are known to be elusive, e.g., having a clique of a certain size or a coloring with a certain number of color classes (Bollobás [3]) or being planar (Best, van Emde Boas, and Lenstra [2]); see [4, 14, 18] for more examples. On the other hand, it is well known that there exist nontrivial graph properties that need only $O(n)$ questions; see [2, 4].

Aanderaa and Rosenberg conjectured [2] that there exist some $\gamma > 0$ such that the complexity of every nontrivial monotone graph property (i.e., a property preserved under deleting edges) is at least $\gamma n^2$. This conjecture has been proved by Rivest and Vuillemin [12] for $\gamma = \frac{1}{16}$. The value of $\gamma$ has been improved over the years. Currently the largest value of $\gamma$ for which the conjecture of Aanderaa and Rosenberg is known to be true is $\frac{1}{4} - o(1)$. This result was established by Kahn, Saks, and Sturtevant [9]. A sharpened version of the Aanderaa–Rosenberg conjecture is due to Karp [13]. He conjectures that every nontrivial monotone graph property is elusive; i.e., he conjectures that $\gamma = \frac{1}{2} - o(1)$ holds.

[*]Received by the editors April 29, 2003; accepted for publication (in revised form) March 24, 2004; published electronically October 8, 2004.
http://www.siam.org/journals/sicomp/34-1/42679.html

[†]Humboldt-Universität zu Berlin, Institut für Informatik, Unter den Linden 6, D-10099 Berlin, Germany (hougardy@informatik.hu-berlin.de).
[‡]Konrad-Zuse-Zentrum für Informationstechnik Berlin (ZIB), Takustr. 7, D-14195 Berlin, Germany (wagler@zib.de).

Karp's conjecture has been verified for several graph properties, some of which have already been mentioned above. The most general result in this area is due to Kahn, Saks, and Sturtevant [9], who proved Karp's conjecture for all $n$ that are prime powers.

Karp's conjecture is a very challenging open problem. But even more challenging is the problem of getting a complete characterization of all graph properties that are elusive. Currently we still seem to be far away from even formulating a reasonable conjecture. Recent progress in this direction has been made by Chakrabarti, Khot, and Shi [6], who proved that any minor closed graph property (which need not be monotone) is elusive. The subject of the present paper is to prove elusiveness for some other nonmonotone graph property.

THEOREM 1.1. *Perfectness is an elusive graph property.*

Perfectness is a property which is not monotone but hereditary (preserved under deleting nodes) and concerned with the relation of maximum cliques and optimal colorings. Perfect graphs behave nicely from an algorithmic point of view [8] and have interesting relationships to surprisingly many other fields of scientific enquiry [11].

Berge [1] proposed calling a graph $G$ *perfect* if, for each of its (node-)induced subgraphs $G' \subseteq G$, the chromatic number equals the clique number (i.e., if we need as many stable sets to cover all nodes of $G'$ as a maximum clique of $G'$ has nodes). This means that identifying one induced imperfect subgraph would enable Player A to make the final decision: the graph in question is not perfect. For that, so-called *minimally imperfect graphs* are of particular interest. These are imperfect graphs, and all of their proper induced subgraphs are perfect. As was proved recently by Chudnovsky et al. [7], the only minimally imperfect graphs are chordless odd cycles of length $\geq 5$, termed *odd holes*, and their complements, called *odd antiholes*. This result was conjectured by Berge in 1960 and is called the strong perfect graph theorem. In our proof for Theorem 1.1 we do not need to rely on this deep result, nor would it help to simplify our proofs.

Player B has to answer in such a way that no minimally imperfect induced subgraph appears until Player A asks the last question but that the last answer can create a minimally imperfect induced subgraph.

The odd hole of length five is the smallest imperfect graph. Hence, the cases with $n \leq 4$ nodes are trivial: Player A knows without asking any question that the studied graph is perfect. In order to show that perfectness is an elusive graph property we have to treat the nontrivial cases $n \geq 5$.

The idea for providing a strategy to Player B is as follows. Find perfect graphs such that you cannot reach another perfect graph by deleting or adding one edge. We call an edge $e$ of a perfect graph $G$ *critical* if $G - e$ is imperfect. Analogously, we call an edge $e$ not contained in a perfect graph $G$ *anticritical* if $G + e$ is imperfect. A perfect graph $G$ is *critical* if $G$ has only critical edges. The complement of a critically perfect graph is again perfect (due to Lovász [10]) and has the property that adding an edge not contained in the graph so far yields an imperfect graph. We call the complements of critically perfect graphs *anticritically perfect.* We look for *bicritically perfect graphs*, which are both critically *and* anticritically perfect: the deletion and addition of an *arbitrary* edge yields an imperfect graph.

If there exists a bicritically perfect graph $G_n$, then Player B has only to answer all but the last question "$ij \in E$?" of Player A as in $G_n$. That is, Player B has only to apply the following strategy for graphs on $n$ nodes.

STRATEGY 1. *Let $G_n$ be a bicritically perfect graph on $n$ nodes.*

*For questions* 1 *to* $\binom{n}{2} - 1$*: Answer "*$ij \in E$*?" with* YES *if* $ij \in E(G_n)$*,* NO *otherwise.*

Then no induced imperfect subgraph appears during the first $\binom{n}{2} - 1$ questions, and the answer to the last question yields the following decision:

$$\text{Answer "}ij \in E\text{?" with } \begin{cases} \text{YES} & \text{if } ij \in E(G_n); \text{ then the graph is perfect.} \\ \text{NO} & \text{if } ij \in E(G_n); \text{ then the graph is imperfect.} \\ \text{YES} & \text{if } ij \notin E(G_n); \text{ then the graph is imperfect.} \\ \text{NO} & \text{if } ij \notin E(G_n); \text{ then the graph is perfect.} \end{cases}$$

In order to prove Theorem 1.1, our task is as follows.

PROBLEM 1. *Find, for as many* $n$ *as possible, a bicritically perfect graph* $G_n$ *on* $n$ *nodes.*

A first step towards solving Problem 1 was a computer search enumerating which perfect graphs on up to 10 nodes are critically perfect.

THEOREM 1.2. *No critically perfect graphs with fewer than* 9 *nodes exist. On* 9*,* 10*, and* 11 *nodes there are precisely* 3*,* 10*, and* 52 *critically perfect graphs.*

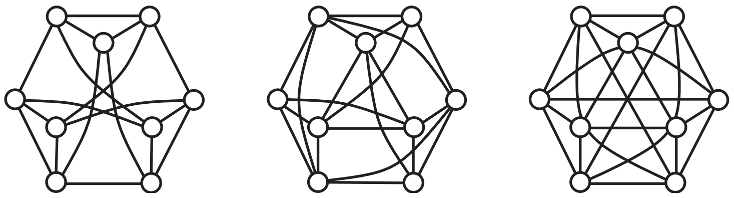


FIG. 1.1. *The three smallest critically perfect graphs.*

Clearly, Theorem 1.2 remains true if "critically perfect" is replaced by "anticritically perfect." Figure 1.1 shows the three critically perfect graphs on nine nodes. The first graph, $G_9$, is self-complementary and, therefore, also anticritical. That is, it is our first example of a bicritically perfect graph. The other two graphs are not anticritical, but only their complements are. Each of the critically perfect graphs with ten nodes is not anticritical (see the next section). This means particularly that there are no bicritically perfect graphs $G_n$ with $n \leq 8$ and $n = 10$. In section 2, we present a technique of constructing examples of bicritically perfect graphs based on the characterization of critically and anticritically perfect line graphs. In section 3, we apply the knowledge from the previous section to construct bicritically perfect graphs $G_n$ if $n \geq 12$. Section 4 provides a slightly different strategy for the cases $n = 10, 11$.

The cases $5 \leq n \leq 8$ are treated as follows. The odd hole $C_5$ is the *only* imperfect graph on five nodes (note: the $C_5$ is self-complementary, and hence so is the odd antihole on five nodes). Consequently, one cannot reach another imperfect graph from the $C_5$ by deleting or adding one edge. Thus, the $C_5$ is bicritically imperfect and Strategy 1 does also work for $n = 5$ when choosing $G_n = C_5$. For $6 \leq n \leq 8$ there is no bicritically imperfect graph with $n$ nodes. In order to treat these cases we do not provide an explicit strategy, but we show in section 5 that there *exists* a strategy: we prove elusiveness for $6 \leq n \leq 8$ with the help of a result of Rivest and Vuillemin [12] by using a parity argument and doing some computer searches.

In summary, we show the existence of a strategy in all nontrivial cases $n \geq 5$ which proves Theorem 1.1: *perfectness is an elusive graph property.*

**2. Bicritically perfect line graphs.** This section provides a construction for bicritically perfect line graphs established in [17]. We obtain the *line graph* $L(F)$ of a graph $F$ by taking the edges of $F$ as nodes of $L(F)$ and joining two nodes of $L(F)$ by an edge iff the corresponding edges of $F$ are incident. It is well known [15] that the line graph $L(F)$ of a graph $F$ is perfect iff $F$ is *line-perfect*, i.e., iff $F$ does not contain any odd cycle of length at least 5 as a (not necessarily induced) subgraph.

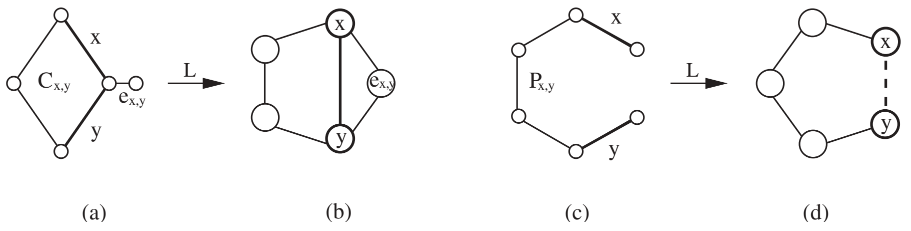


FIG. 2.1. *Definition of H-pairs and A-pairs.*

In order to obtain critical and anticritical edges in $L(F)$, we define two structures in $F$.

We say that two incident edges $x$ and $y$ form an *H-pair* in $F$ if there is an edge $e_{x,y}$ different from $x$ and $y$ incident to the common node of $x$ and $y$ and if there is an even cycle $C_{x,y}$ containing $x$ and $y$ but only one endnode of $e_{x,y}$ (see Figure 2.1(a)). $L(C_{x,y})$ is an even hole, and the node in $L(F)$ corresponding to $e_{x,y}$ has precisely two neighbors on $L(C_{x,y})$, namely, $x$ and $y$ (see Figure 2.1(b)).

Two nonincident edges $x$ and $y$ are called an *A-pair* if they are the endedges of an odd path $P_{x,y}$ with length at least five (see Figure 2.1(c)). $L(P_{x,y})$ is an even, chordless path of length at least four with endnodes $x$ and $y$ (see Figure 2.1(d)).

It is straightforward that deleting and adding the edge $xy$ in $L(C_{x,y} \cup e_{x,y})$ and $L(P_{x,y})$, respectively, yields an odd hole. Consequently, if $L(F)$ is intended to be critically (anticritically) perfect, it is sufficient to obtain that *every* pair of incident (nonincident) edges in $F$ forms an H-pair (A-pair). We define a graph with at least two incident (nonincident) edges to be an *H-graph* (*A-graph*) if each pair of incident (nonincident) edges forms an H-pair (A-pair).

Obviously, the line graph $L(F)$ of any bipartite H-graph (A-graph) is critically (anticritically) perfect. A line graph $L(F)$ is bicritically perfect iff the graph $F$ is a bipartite A- and H-graph [16]. The three smallest critically perfect graphs are the complements of the line graphs of the three bipartite A-graphs presented in Figure 2.2. $A_1$ is also an H-graph, and hence $L(A_1)$ is bicritical (it is in fact self-complementary). Furthermore, $A_1$ is the only bipartite H-graph with 3 nodes in each color class. If there are 4 nodes in one color class, then an H-graph has at least 12 edges since it has minimum degree 3 by definition. Hence, the second smallest H-graph admits 12 edges and there cannot be any bicritically perfect *line* graph on 10 or 11 nodes. By complete enumeration we proved that bicritically perfect graphs on 10 or 11 nodes do not exist at all.

The following sufficient condition for a bipartite graph $F$ to be an H-graph *and* an A-graph is established in [17].

LEMMA 2.1 (see [17]). *Every simple,* 3*-connected, bipartite graph is an H-graph as well as an A-graph.*

*Proof.* Let $F = (A \cup B, E)$ be a simple, 3-connected, bipartite graph. First, consider two arbitrary incident edges $ab_1, ab_2$ of $F$ with $a \in A$ and $b_1, b_2 \in B$. We

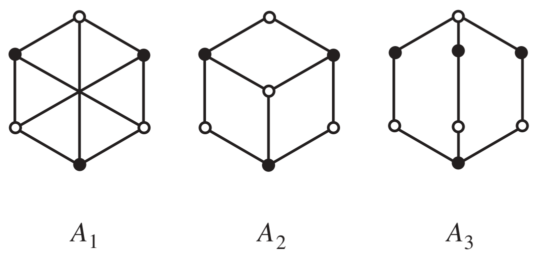


FIG. 2.2. *The three smallest bipartite A-graphs.*

show that $ab_1$ and $ab_2$ form an H-pair in $F$. Since $F$ is 3-connected, there is a third node $b_3 \neq b_1, b_2$ adjacent to $a$ and $F - \{a, b_3\}$ is still connected. In particular, $b_1$ and $b_2$ are linked by a path $P$ in $F - \{a, b_3\}$. Hence, we obtain $C_{ab_1,ab_2} = P \cup \{b_2a, ab_1\}$ and $e_{ab_1,ab_2} = ab_3$; i.e., $ab_1$ and $ab_2$ form an H-pair in $F$.

Now, consider two nonincident edges $a_1b_1$ and $a_2b_2$ of $F$ with $a_i \in A$, $b_i \in B$. We show that $a_1b_1$ and $a_2b_2$ form an A-pair in $F$. Since $F$ is 3-connected, there are internally disjoint, odd $(a_1, b_2)$-paths $P_1, \ldots, P_k$ with $k \geq 3$. At most two paths among $P_1, \ldots, P_k$ can contain $a_2$ or $b_1$. Without loss of generality, let $P_3, \ldots, P_k$ be $a_2, b_1$-free. If there is a path $P_i$ with $3 \leq i \leq k$ of length $\geq 3$, then $b_1P_ia_2$ is the studied path $P_{a_1b_1,a_2,b_2}$. Otherwise, the only $a_2, b_1$-free $(a_1, b_2)$-path is the edge $a_1b_2$. We obtain $k = 3$ and let $b_1 \in P_1$, $a_2 \in P_2$. By the 3-connectivity of $F$ again, there must be an $a_1, b_2$-free $(a_2, b_1)$-path $Q$ of odd length $\geq 3$, and we get $P_{a_1b_1,a_2,b_2} = b_2Qa_1$. Otherwise, if the edge $a_2b_1$ were the only $a_1, b_2$-free $(a_2, b_1)$-path, $\{a_1, a_2\}$ would be a cutset of size two, separating the nodes of $P_1$ between $b_1$ and $b_2$ from the nodes of $P_2$ between $a_1$ and $a_2$—a contradiction, as $F$ is 3-connected. □

*Remark* 2.2. Note that duplicating edges preserves the property of being an A-graph (since no new pair of nonincident edges occurs), while it does not preserve the property of being an H-graph (since parallel edges are incident but never form an H-pair).

## 3. Construction of the graphs $G_n$ for $n \geq 12$.

In order to treat Problem 1, this section is intended to present a bicritically perfect graph $G_n$ for each $n \geq 12$. Lemma 2.1 ensures that $L(F)$ is bicritically perfect whenever $F$ is a simple, 3-connected, bipartite graph. Hence we will construct simple, 3-connected, bipartite graphs $F_n$ with $n \geq 12$ edges to obtain the studied bicritically perfect graphs $G_n = L(F_n)$ on $n \geq 12$ nodes. Consider the graphs $F_{3k} = (A \cup B, E_1 \cup E_2)$ with $k \geq 3$ and

$$\begin{aligned} A &= \{1, 3, \ldots, 2k-1\}, \\ B &= \{2, 4, \ldots, 2k\}, \\ E_1 &= \{ii+1 : 1 \leq i \leq 2k\}, \\ E_2 &= \{ii+3 : i \in A\}, \end{aligned}$$

where all indices are taken modulo $2k$. The three smallest examples of graphs $F_{3k}$ for $k \in \{3, 4, 5\}$ are shown in Figure 3.1 (note $A_1 = F_9$). $F_{3k}$ is an even cycle $(A \cup B, E_1)$ on its $2k$ nodes with $k$ chords $E_2$ outgoing from a node in $A$ with odd index and ending in a node in $B$ with even index. Thus, the graphs $F_{3k}$ are bipartite and simple by construction. We have to show that they are 3-connected.

LEMMA 3.1. *The graphs $F_{3k}$ are 3-connected for $k \geq 3$.*

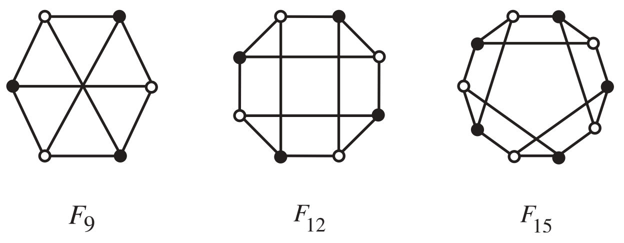


FIG. 3.1. *The graphs $F_{3k}$ with $k = 3, 4, 5$.*

*Proof.* We have to show that the graph obtained from $F_{3k}$ by removing two arbitrary nodes $i$ and $j$ is still connected. Let $i < j$. Recall that $F_{3k} = (A \cup B, E_1 \cup E_2)$ has a Hamilton cycle $C = (A \cup B, E_1)$ and additional chords $ii + 3 \in E_2$ with $i \in A$ odd, $i + 3 \in B$ even. If $i$ and $j$ are neighbors on $C$ (i.e., $j = i + 1$), then the remaining nodes $i + 2 = j + 1, \ldots, 2k, 1, \ldots, i - 1$ of $F_{3k}$ are connected by a path with edges in $E_1$. Otherwise, removing $i$ and $j$ divides the cycle $C$ into two paths, $P_1 = i + 1, \ldots, j - 1$ and $P_2 = j + 1, \ldots, 2k, 1, \ldots, i - 1$. It is easy to see that there is always an edge $e \in E_2$ which connects $P_1$ and $P_2$: If $i$ is odd, then $i + 1$ is even and $i - 2i + 1 \in E_2$. We have $e = i - 2i + 1$ as the studied edge connecting $P_1$ and $P_2$ if $i - 2$ is a node of $P_2$ or else $V(P_2) = \{i - 1\}$ and $j = i - 2$ holds. But then $i - 4$ is a node of $P_1$ (since $k \geq 3$) and we obtain $e = i - 4i - 1$ (all indices are taken modulo $2k$). Analogously, if $i$ is even, then $i - 1i + 2 \in E_2$. We have $e = i - 1i + 2$ if $i + 2$ is a node of $P_1$ or else $V(P_1) = \{i + 1\}$ and $j = i + 2$. But then $i + 4$ is a node of $P_2$ (by $k \geq 3$ again) and we get $e = i + 1i + 4$. Hence, the graph obtained from $F_{3k}$ by removing two arbitrary nodes is still connected. ☐

Thus, we can choose $G_n$ as the line graph of $F_n$ whenever $n = 3k, k \geq 3$, by Lemma 2.1. To close the gaps with $n = 3k + 1, 3k + 2$ for $k \geq 4$ we use the following immediate consequence of Lemma 2.1.

LEMMA 3.2. *If $F = (A \cup B, E)$ is a simple, 3-connected, bipartite graph and $ab \notin E$ with $a \in A, b \in B$, then $F + ab$ is a simple bipartite A- and H-graph.*

Thus, we obtain the studied bipartite A- and H-graphs $F_n$ for $n = 3k + 1$ and $n = 3k + 2$ if $k \geq 4$ by adding one and two edges, respectively, to $F_{3k}$ such that the resulting graph is simple and bipartite. This is possible for each $F_{3k}$ with $k \geq 4$ (but not for the complete bipartite graph $F_9$). We obtain, therefore, the studied bicritically perfect graphs $G_n = L(F_n)$ for $n \geq 12$ and can apply Strategy 1 for all cases with $n \geq 12$ nodes.

## 4. Construction of the graphs $G_n$ for $n = 10, 11$.

As mentioned in section 2 there are no bicritically perfect graphs on 10 or 11 nodes. Therefore we construct bipartite A-graphs with 10 and 11 edges which are as close to H-graphs as possible.

Duplicating an arbitrary edge of $F_9 = A_1$ yields the graph $F_{10}$ shown in Figure 4.1. $F_{10}$ is an A-graph but not an H-graph by Remark 2.2. However, $L(F_{10})$ has only one noncritical edge, namely, the edge connecting the nodes that correspond to the parallel edges of $F_{10}$. Next, the bipartite graph $F_{11}$ in Figure 4.1 can be obtained by adding two edges to the A-graph $A_2$ from Figure 2.2. It is easy to check that $F_{11}$ is an A-graph and that the two edges incident to the only node of degree two in $F_{11}$ form the only non-H-pair. $L(F_{11})$ is anticritically perfect and all edges but one are critical, too.

Let us call a graph $G$ *almost bicritically perfect* if $G$ is anticritically perfect and all

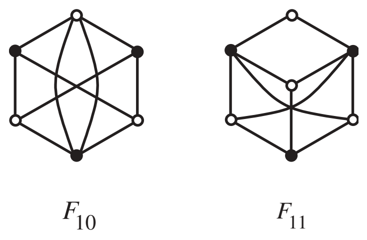


FIG. 4.1. *The graphs $F_{10}$ and $F_{11}$.*

edges but one are critical. Then we slightly modify Strategy 1 for almost bicritically perfect graphs as follows.

STRATEGY 2. *Let $G_n$ be an almost bicritically perfect graph on $n$ nodes and let $uv$ be its only noncritical edge. For question* 1*: Answer "$ij \in E$?" with* YES*; number the nodes of $G_n$ s.t. $i = u$, $j = v$. For questions* 2 *to* $\binom{n}{2} - 1$*: Answer "$ij \in E$?" with YES if $ij \in E(G_n)$,* NO *otherwise.*

Then no imperfect subgraph appears during the first $\binom{n}{2} - 1$ questions, and the answer to the last question yields the decision again. Since $L(F_{10})$ and $L(F_{11})$ are almost bicritically perfect graphs by construction, we choose $G_{10} = L(F_{10})$ and $G_{11} = L(F_{11})$ and apply Strategy 2.

**5. The remaining cases $6 \leq n \leq 8$.** To prove that perfectness is an elusive graph property for $6 \leq n \leq 8$ we use a parity argument due to Rivest and Vuillemin. In [12] they proved the following: If a property $\mathcal{P}$ is *not* elusive for graphs on $n$ nodes, then the number $G(\mathcal{P}, n, \text{even})$ of labeled graphs on $n$ nodes with property $\mathcal{P}$ having an even number of edges equals the number $G(\mathcal{P}, n, \text{odd})$ of labeled graphs on $n$ nodes with property $\mathcal{P}$ that have an odd number of edges. In particular, $G(\mathcal{P}, n, \text{even}) \neq G(\mathcal{P}, n, \text{odd})$ implies that $\mathcal{P}$ *is* elusive for graphs on $n$ nodes. In Table 5.1 we show the numbers $G(\mathcal{P}, n, \text{even})$ and $G(\mathcal{P}, n, \text{odd})$ for perfect graphs on $6 \leq n \leq 8$ nodes, which we calculated with the help of a computer program. For the calculation we used a very simple brute force approach: We generated all graphs on up to 8 nodes. Such a graph is perfect iff it does not contain a $C_5$, a $C_7$, or a complement of a $C_7$ as an induced subgraph. Now one simply has to count how many of these graphs have an even, respectively, odd number of edges. As one can see from Table 5.1, for $n = 8$ perfectness is an elusive graph property, as the values in columns 3 and 4 differ.

TABLE 5.1
*The number of perfect graphs with an even, respectively, odd number of edges.*

| $n$ | # perfect graphs | $G(\mathcal{P}, n, \text{even})$ | $G(\mathcal{P}, n, \text{odd})$ |
|---|---|---|---|
| 6 | 30824 | 15412 | 15412 |
| 7 | 1741616 | 870808 | 870808 |
| 8 | 174494128 | 87264704 | 87229424 |

For $n = 6$ and $n = 7$ we apply an extension of the previously used argument. If perfectness is not elusive for graphs on $n$ nodes, then it is also not elusive for the graphs containing one fixed edge, say $ij$. Therefore the number of labeled perfect graphs on $n$ nodes which contain the edge $ij$ and have an even number of edges must equal the number of these graphs with an odd number of edges. The last two columns in Table 5.2 show these numbers for $n = 6$ and $n = 7$. Note that they add up to half the number of labeled perfect graphs, as the complement of a perfect graph is

TABLE 5.2
*The number of perfect graphs with a fixed edge and an even, respectively, odd number of edges.*

| $n$ | # perfect graphs | $G(\mathcal{P}, n, \text{even})$, $ij \in E$ | $G(\mathcal{P}, n, \text{odd})$, $ij \in E$ |
|---|---|---|---|
| 6 | 30824 | 7712 | 7700 |
| 7 | 1741616 | 435284 | 435524 |

again perfect [10]. As the numbers in columns 3 and 4 are different in both cases, this finishes the proof that perfectness is elusive for $n = 6, 7, 8$.

## 6. Summary.

In order to figure out whether perfectness is an elusive graph property, we used the following as our main idea: Find, for as many numbers $n$ of nodes as possible, a bicritically perfect graph $G_n$ (Problem 1). Since one cannot reach another perfect graph from $G_n$ by deleting or adding one edge, there is a simple strategy for Player B in that case: Answer all but the last question as in the bicritically perfect graph $G_n$ (Strategy 1). We constructed bicritically perfect graphs $G_n$ with $n = 9$ and $n \geq 12$ (section 3) and almost bicritically perfect graphs $G_{10}$ and $G_{11}$ (section 4) where a slightly different strategy has to be used (Strategy 2). Moreover, the $C_5$ is bicritically imperfect and Strategy 1 does also work for $n = 5$ with choosing $G_n = C_5$. Consequently, our main idea works for $n = 5$ and for all cases with $n \geq 9$ nodes. We used a parity argument from [12] in order to show the desired result for the remaining cases with $6 \leq n \leq 8$ nodes (section 5).

In summary, we showed the *existence* of a strategy for Player B in all nontrivial cases $n \geq 5$, which finally proves Theorem 1.1: *Perfectness is an elusive graph property.*

**Acknowledgment.** The authors are grateful to Günter M. Ziegler for drawing their attention to the result of Rivest and Vuillemin [12].

## REFERENCES

[1] C. Berge, *Färbung von Graphen, deren sämtliche bzw. deren ungerade Kreise starr sind*, Math.-Natur. Reihe, 10 (1961), pp. 114–115.
[2] M. R. Best, P. van Emde Boas, and H. W. Lenstra, *A Sharpened Version of the Aanderaa-Rosenberg Conjecture*, Technical Report ZW 30/74, Mathematisch Centrum Amsterdam, Afd. Zuivere Wisk., 1974.
[3] B. Bollobás, *Complete subgraphs are elusive*, J. Combinatorial Theory Ser. B, 21 (1976), pp. 1–7.
[4] B. Bollobás, *Extremal Graph Theory*, Academic Press, London, New York, 1978.
[5] E. Boros, V. Gurvich, and S. Hougardy, *Recursive generation of partitionable graphs*, J. Graph Theory, 41 (2002), pp. 259–285.
[6] A. Chakrabarti, S. Khot, and Y. Shi, *Evasiveness of subgraph containment and related properties*, SIAM J. Comput., 31 (2002), pp. 866–875.
[7] M. Chudnovsky, N. Robertson, P. D. Seymour, and R. Thomas, *The Strong Perfect Graph Theorem*, manuscript, 2003.
[8] M. Grötschel, L. Lovász, and A. Schrijver, *Geometric Algorithms and Combinatorial Optimization*, Springer-Verlag, Berlin, 1988.
[9] J. Kahn, M. Saks, and D. Sturtevant, *A topological approach to evasiveness*, Combinatorica, 4 (1984), pp. 297–306.
[10] L. Lovász, *Normal hypergraphs and the perfect graph conjecture*, Discrete Math., 2 (1972) pp. 253–267.
[11] B. Reed and J. Ramirez-Alfonsin, eds., *Perfect Graphs*, Wiley, Chichester, UK, 2001.
[12] R. L. Rivest and J. Vuillemin, *On recognizing graph properties from adjacency matrices*, Theoret. Comput. Sci., 3 (1976/77) pp. 371–384.
[13] A. L. Rosenberg, *On the time required to recognize properties of graphs: A problem*, SIGACT News, 5 (1973), pp. 15–16.

[14] E. Triesch, *Some results on elusive graph properties*, SIAM J. Comput., 23 (1994), pp. 247–254.
[15] L. E. Trotter Jr., *Line perfect graphs*, Math. Programming, 12 (1977), pp. 255–259.
[16] A. Wagler, *On critically perfect graphs*, J. Graph Theory, 32 (1999), pp. 394–404.
[17] A. Wagler, *Critical Edges in Perfect Graphs*, Ph.D. thesis, TU Berlin, 2000.
[18] A. C.-C. Yao, *Monotone bipartite graph properties are evasive*, SIAM J. Comput., 17 (1988), pp. 517–520.

SIAM J. COMPUT.
Vol. 34, No. 1, pp. 118–169

# ALMOST PERFECT LATTICES, THE COVERING RADIUS PROBLEM, AND APPLICATIONS TO AJTAI'S CONNECTION FACTOR[*]

DANIELE MICCIANCIO[†]

**Abstract.** Lattices have received considerable attention as a potential source of computational hardness to be used in cryptography, after a breakthrough result of Ajtai [in *Proceedings of the* 28*th Annual ACM Symposium on Theory of Computing*, Philadelphia, PA, 1996, pp. 99–108] connecting the average-case and worst-case complexity of various lattice problems. The purpose of this paper is twofold. On the expository side, we present a rigorous self-contained proof of results along the lines of Ajtai's seminal work. At the same time, we explore to what extent Ajtai's original results can be quantitatively improved. As a by-product, we define a random class of lattices such that computing short nonzero vectors in the class with nonnegligible probability is at least as hard as approximating the length of the shortest nonzero vector in *any* $n$-dimensional lattice within worst-case approximation factors $\gamma(n) = n^3\omega(\sqrt{\log n \log\log n})$. This improves previously known best connection factor $\gamma(n) = n^{4+\epsilon}$ [J.-Y. Cai and A. P. Nerurkar, in *Proceedings of the* 38*th Annual IEEE Symposium on Foundations of Computer Science*, Miami Beach, FL, 1997, pp. 468–477]. We also show how our reduction implies the existence of collision resistant cryptographic hash functions based on the worst-case inapproximability of the shortest vector problem within the same factors $\gamma(n) = n^3\omega(\sqrt{\log n \log\log n})$.

In the process we distill various new lattice problems that might be of independent interest, related to the covering radius, the bounded distance decoding problem, approximate counting of lattice points inside convex bodies, and the efficient construction of lattices with good geometric and algorithmic decoding properties. We also show how further investigation of these new lattice problems might lead to even stronger connections between the average-case and worst-case complexity of the shortest vector problem, possibly leading to connection factors as low as $\gamma(n) = n^{1.5}\omega(\log n)$.

**Key words.** point lattices, shortest vector problem, average-case complexity, covering radius, cryptography, hash functions, almost perfect lattices, closest vector problem with preprocessing

**AMS subject classifications.** 52C07, 52C17, 68Q17, 68W25, 68R99, 94B75, 11P21

**DOI.** 10.1137/S0097539703433511

## 1. Introduction.

It has long been realized that the relevant notion of hardness in cryptography is *average-case* hardness: if the key of a cryptographic function is chosen at random, then no probabilistic polynomial time algorithm can break the scheme with nonnegligible probability. In the past few years, computational problems on point lattices have attracted considerable interest for their potential cryptographic applications because of the following remarkable discovery of Ajtai [2]: a certain natural computational problem (namely, finding small nonzero solutions to a suitably chosen random system of homogeneous linear equations) is at least as hard *on the average* as the *worst-case* instance of various lattice problems, e.g., approximating the length of the shortest nonzero vector in a lattice within a worst-case factor $\gamma(n)$

[*]Received by the editors August 15, 2003; accepted for publication (in revised form) March 28, 2004; published electronically October 28, 2004. Preliminary versions of this paper appeared in the proceedings of STOC 2002 and CCC 2002 with the title *Improved cryptographic hash functions with worst-case/average-case connection* [35], and in Chapter 8 of the book *Complexity of Lattice Problems: A Cryptographic Perspective* [36]. This research was supported in part by NSF Career Award CCR-0093029 and a Sloan Research Fellowship. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author and do not necessarily reflect the views of the National Science Foundation.

http://www.siam.org/journals/sicomp/34-1/43351.html

[†]Computer Science and Engineering Department, University of California at San Diego, 9500 Gilman Dr., Mail code 0114, La Jolla, CA, 92093 (daniele@cs.ucsd.edu).

polynomial in the dimension $n$ of the lattice. This immediately gives provably secure cryptographic functions based on the conjectured *worst-case* intractability of lattice problems.[1] Moreover, since the set of integer solutions of a linear system forms a lattice, the result in [2] can also be regarded as a connection between the complexity of finding short nonzero vectors in suitably chosen random lattices *on the average*,[2] and the complexity of approximating the length of the shortest nonzero vector (as well as solving various other lattice problems) in any lattice *in the worst case*.

We remark that building cryptographic functions that are as hard to break as the *worst-case* instance of the underlying mathematical problem is especially important in the case of lattices because lattice approximation algorithms (like the Lenstra–Lenstra–Lovász (LLL) algorithm [27]) are believed to perform much better on the average than their worst-case theoretical upper bounds. Moreover, since lattice problems get easier and easier as the approximation factor $\gamma(n)$ increases, determining the smallest worst-case inapproximability factor $\gamma(n)$ that implies the average-case hardness of solving Ajtai's random equations is both a theoretically interesting and a practically important problem, and it has been the subject of subsequent work by Cai and Nerurkar [11] and Micciancio [35] in a preliminary version of this paper.

*Our contribution.* The purpose of this paper is twofold. First, we give a full, self-contained proof of Ajtai's original result [2] and a detailed account of all relevant techniques introduced in subsequent improvements by Cai and Nerurkar [11]. Previous papers [2, 11] appeared only in the form of extended abstracts or technical reports that left to the reader the burden of reconstructing the details of many technical steps. In this paper we develop a number of elementary, still useful, general techniques that allow us both to cover all the steps in great detail, and at the same time also offer a cleaner high level picture of the proof. Second, we explore to what extent the worst-case inapproximability factors $\gamma(n)$ for lattice problems (shown to imply the average-case hardness of solving random linear equations in [2, 11]) can be further reduced. In the process, we introduce and start investigating various new lattice problems that might be of independent interest and that are discussed in more detail in the following subsections. These technical contributions are summarized as follows. On the average-case complexity side, we introduce a kind of lattice (that we call *almost perfect* in analogy with perfect codes), and use lattices of this kind to define a *new random class* of linear equations such that finding small solutions on the average is potentially harder than in the random class proposed by Ajtai. On the worst-case complexity side, we consider various *new lattice problems*, like approximating the *covering radius* of a lattice. Using these new problems, we are able to improve the connection factor for the shortest vector problem (SVP) established in [2, 11]. Specifically, we show that finding small solutions to our random equations (with nonnegligible probability) is at least as hard as the *worst-case* instance of

(i) approximating the length of the shortest nonzero vector in any $n$-dimensional

[1]In particular, Ajtai [2] showed that if no algorithm can efficiently approximate the length of the shortest nonzero vector in any $n$-dimensional lattice within (worst-case) polynomial approximation factors $\gamma(n) = n^{O(1)}$, then *one-way functions* exist. Subsequently, Goldreich, Goldwasser, and Halevi [18] observed that under essentially the same assumptions as Ajtai's, one can prove the existence of *collision resistant hash functions*, a particularly useful kind of one-way function family with many applications in cryptography.

[2]For clarity, in the rest of the paper we always refer to this average-case problem as "finding small solutions to random equations," while lattices are used only to describe worst-case problems. However, we remark that the two formulations are equivalent, and all results discussed in this paper can be regarded as connections between the average-case and worst-case complexity of (different) lattice problems.

lattice within a factor $\gamma(n) = \tau(n) \cdot n^{2.5}\omega(\log n)$, where $\tau(n) \in [1, \sqrt{n}]$ is a parameter that depends on the almost perfect lattices used in the construction and $\omega(\log n)$ is an arbitrary superlogarithmic function.

Even for $\tau(n) = \sqrt{n}$ this improves the connection factor $\gamma(n) = n^{4+\epsilon}$ of [11] by more than a factor $n$. (See section 1.3 for details about the results of [11].) We remark that function $\tau(n)$ is a parameter that depends on the construction of the random equations, and it equals $\sqrt{n}$ in Ajtai's construction as studied in [2, 11]. In this paper we give a better construction showing that smaller values of $\tau(n)$ are possible. The improvement we present is quantitatively modest (namely, $\tau(n) = \sqrt{n \log\log n / \log n}$) but qualitatively interesting, as it shows that Ajtai's class of random equations is not necessarily optimal, and there is room to hope that more substantial improvements are possible.

We also relate the average-case complexity of computing small solutions to our random equations to other lattice problems, like the *worst-case* instances of the following:

(ii) computing maximal sets of linearly independent vectors that are within a factor $\gamma(n) = \tau(n) \cdot n^2 \cdot \omega(\log n)$ from the shortest,[3]

(iii) approximating within a factor $\gamma(n) = \tau(n) \cdot n^2 \cdot \omega(\log n)$ the covering radius of any $n$-dimensional lattice, i.e., the maximum possible distance $\rho(\mathcal{L}(\mathbf{B})) = \max_{\mathbf{t}} \operatorname{dist}(\mathbf{t}, \mathcal{L}(\mathbf{B}))$ where $\mathbf{t}$ ranges over the entire space spanned by $\mathbf{B}$,

(iv) finding, given an $n$-dimensional lattice basis $\mathbf{B}$ and a target point $\mathbf{t}$, a lattice point whose distance from the target $\mathbf{t}$ is at most $\gamma(n) = \tau(n) \cdot n^2 \cdot \omega(\log n)$ times the covering radius of the lattice.

Even for $\tau(n) = \sqrt{n}$, the first relation improves previously known best connection factor $n^{3+\epsilon}$ of [11] by more than $\sqrt{n}$. (See section 1.3 for details about the results of [11].) The other two relations are the first results explicitly connecting the complexity of finding small solutions to random equations to the covering radius problem (CRP). Although neither problem has been previously considered in computational complexity, they are both natural computational problems on lattices that might be of independent interest. The last problem is a "guaranteed distance" variant of the well-studied closest vector problem (CVP), where the error, instead of being measured with respect to the distance of the given target, is measured with respect to the worst-case distance over all possible target vectors.

All our results are obtained as corollaries to a main theorem that shows that finding small solutions to our random equations is at least as hard as the worst-case instance of finding maximal sets of linearly independent vectors of length at most $\gamma(n) = \tau(n)\sqrt{n} \cdot \omega(\log n)$ times a new lattice invariant that we call the *generalized uniform radius*. Notice how this factor is extremely small: depending on the value of $\tau(n)$, $\gamma(n)$ can be as small as $\sqrt{n} \cdot \omega(\log n)$. This suggests that further investigation of almost perfect lattices and the connection between the uniform radius and other lattice invariants might lead to even stronger connections between the average-case and worst-case complexity of computing short lattice vectors. In particular, we conjecture that there exist random classes of linear equations such that finding small solutions on the average is at least as hard as approximating the length of the shortest nonzero vector in any $n$-dimensional lattice within a factor $\gamma(n) = n^{1.5} \cdot \omega(\log n)$ in the worst case.

[3]Here, and throughout the rest of the paper, the length of a finite set of (linearly independent) vectors is defined as the maximum length of the vectors in the set.

In the following subsections we give a more detailed description of the new lattice problems introduced in this paper and then review previous work in related areas.

**1.1. New lattice problems.** A lattice is the set of intersection points of a regular, but not necessarily orthogonal, $n$-dimensional grid. Mathematically, it can be described as the set of all integer linear combinations $x_1\mathbf{b}_1 + \cdots + x_n\mathbf{b}_n$ (with $x_1, \ldots, x_n \in \mathbb{Z}$) of a sequence of linearly independent vectors $\mathbf{b}_1, \ldots, \mathbf{b}_n$ in Euclidean space $\mathbb{R}^m$. The simplest example is the *integer lattice* $\mathbb{Z}^n$, i.e., the set of all $n$-dimensional vectors with integer coordinates. Two fundamental constants associated to any lattice are the *packing radius* and the *covering radius*: the packing radius is the largest radius such that (open) spheres centered at distinct lattice points do not intersect, and the covering radius is the smallest radius such that (closed) spheres centered at all lattice points cover the entire space. Equivalently, the *packing radius* can be defined as the largest $r$ such that any (open) sphere of radius $r$ (not necessarily centered around a lattice point) contains *at most one* lattice point. Similarly, the *covering radius* can be defined as the smallest $r$ such that any (closed) sphere of radius $r$ contains *at least one* lattice point. In this paper we introduce a new quantity, the *uniform radius*, defined as the smallest $r$ such that all spheres of radius $r$ contain *approximately the same number* of lattice points. (See section 3 for a formal definition.) For technical reasons, we also introduce a variant of the uniform radius, the *generalized uniform radius*, which considers not only spheres but also arbitrary convex bodies.

Of all these quantities, only the *packing radius* has received considerable attention from a computational complexity point of view. It is easy to see that for any lattice, the packing radius equals half the length of the shortest nonzero lattice vector, so (approximately) computing the packing radius is computationally equivalent to computing the (approximate, within the same approximation factor) length of the shortest nonzero lattice vector. (See section 2.3 for a discussion of the computational complexity of this problem.)

Determining the *covering radius* of a lattice is a classic problem in the geometry of numbers, but so far it has received very little attention from a computational complexity point of view. We suggest that the covering radius is, by itself, an interesting problem to be studied as a potential source of computational hardness. We conjecture that computing the covering radius is NP-hard. We remark that even for the exact version of the problem, no NP-hardness result is currently known. However, the exact solution to the CRP is not even known to be computable in *nondeterministic* polynomial time (NP), and the analogous problem for linear codes is known to be complete for the second level of the polynomial hierarchy [30], a class of problems presumably much harder than NP-complete ones.

The problem of estimating the *(generalized) uniform radius* has been implicitly considered before in connection with vector quantization[4] [29] and volume estimation problems [25] but only for the special case of the integer lattice $\mathbb{Z}^n$ and specific convex bodies (spheres or polyhedra). In this paper we generalize this natural geometric problem to arbitrary lattices and convex bodies and show that the generalized uniform radius is always within a factor $O(n)$ from the covering radius.

**1.2. Almost perfect lattices.** The packing radius and covering radius have been extensively studied in coding theory. *Codes* are sets of strings (called *codewords*) of some fixed length $n$ over a finite alphabet $\Sigma$, with the (Hamming) distance between

[4]Vector quantization is the problem of mapping arbitrary real vectors to a discrete set of points (e.g., the points of a lattice) in such a way that each vector is mapped to a nearby point.

strings measured as the number of positions in which the two strings differ. Similarly to lattices, the packing radius and covering radius of a code are defined as the largest and smallest radii such that the Hamming spheres centered at codewords are disjoint or cover the entire space $\Sigma^n$, respectively. A code is called *perfect* if the packing radius equals the covering radius. In other words, the code is perfect if it is possible to partition the entire space $\Sigma^n$ with equal (Hamming) spheres centered at the codewords. Interestingly, perfect codes are rare but do exist (see [21, section 5]). However, the same is not true for lattices: it is not possible to partition the Euclidean space $\mathbb{R}^n$ with equal spheres of radius bounded away from 0. However, one can attempt to partition the space with almost spherical bodies. Any lattice naturally defines a partition of space into regions, the *Voronoi cells*, obtained by mapping each point in space to the closest lattice point (with ties broken in some standard way). It is easy to see that each Voronoi cell contains an open sphere of radius equal to the packing radius and is completely contained in a closed sphere of radius equal to the covering radius. The covering radius is always at least as large as the packing radius, and the smaller the gap between the two radii, the closer the Voronoi cells are to perfect spheres. We say that a lattice is $\tau$-perfect if the covering radius is at most $\tau$ times the packing radius. We are interested in lattices that are $\tau$-perfect for $\tau > 1$ as small as possible. Notice that the integer lattice $\mathbb{Z}^n$ is $\tau(n)$-perfect for $\tau(n) = \sqrt{n}$, so—is trying to minimize $\tau(n)$—we may assume without loss of generality that $\tau(n) \in [1, \sqrt{n}]$. We say that a sequence of lattices is *almost perfect* if it is $\tau(n)$-perfect for some function $\tau(n) = o(\sqrt{n})$ asymptotically smaller than $\sqrt{n}$. Ideally, we would like to find almost perfect lattices with $\tau(n) = O(1)$ equal to a constant independent of the dimension.

Another fundamental problem in coding theory is the maximum likelihood decoding: given a target point, find the codeword closest to the target. The analogous problem on lattices, called the CVP, is as follows: given a lattice and a target vector, find the lattice point closest to it. Differently from lattices, in coding theory most work has focused on finding efficient decoding algorithms for specific codes, whereas in the CVP the lattice is usually considered part of the input. In this paper, we consider the lattice decoding problem for specific lattices. We say that a lattice is *easily decodable* if there is an efficient algorithm that, on input a target point, outputs the lattice point closest to the target. (Formally, we need to consider a sequence of lattices in higher and higher dimensions. See section 4 for details.) For example, the integer lattice $\mathbb{Z}^n$ is easily decodable: given a target point $\mathbf{y} \in \mathbb{Q}^n$, the closest lattice point is easily found by rounding each coordinate of $\mathbf{y}$ to the closest integer.

The random classes of equations defined in this paper are based on easily decodable $\tau(n)$-perfect lattices, and the smaller $\tau(n)$ is, the harder it is to find small solutions to the random equations. So, it is natural to ask, What is the smallest value of $\tau(n)$ for which we can efficiently build *easily decodable* $\tau(n)$-perfect lattices? It is known [40, 10] that very good almost perfect lattices exist, achieving constant $\tau(n) = O(1)$. Unfortunately, the proofs in [40, 10] do not give an efficient procedure to build and decode these lattices. Various examples of easily decodable lattices are given in [12], but none of them is almost perfect; i.e., they achieve only $\tau(n) = \Theta(\sqrt{n})$. It is natural to ask if almost perfect easily decodable lattices exist at all. In this paper we initiate the study of almost perfect lattices from a computational point of view, and we give the first efficient construction of easily decodable almost perfect lattices with $\tau(n) = O(\sqrt{n \log\log n / \log n})$.

Our almost perfect lattices allow us to slightly improve (by a multiplicative factor $O(\sqrt{\log n / \log\log n})$) the worst-case/average-case connection factor $\gamma(n)$ for all lattice problems considered in this paper. Although not substantial, this improvement in the connection factor is qualitatively important because it shows that there are random classes of linear equations for which finding small solutions is potentially harder than for the random class originally considered by Ajtai. Moreover, it suggests that it might be possible to find even better easily decodable almost perfect lattices that allow us to further reduce the connection factors for all lattice problems considered in this paper by almost $\sqrt{n}$.

**1.3. Related work.** This work directly builds upon techniques of Ajtai [2], Cai and Nerurkar [11] and Goldreich, Goldwasser, and Halevi [18], and it is the final version of [35]. In [2] Ajtai showed[5] that if one can efficiently find small solutions to random linear equations on the average with nonnegligible probability $\delta(n) = 1/n^{O(1)}$, then one can efficiently approximate the length of the shortest nonzero vector in any $n$-dimensional lattice within a (worst-case) polynomial factor $\gamma(n) = n^{O(1)}/\delta(n)$, where the smaller the success probability $\delta(n)$, the larger the approximation factor $\gamma(n)$.[6] Moreover, even for large values of $\delta(n)$ (say, $\delta(n) = 1/2$), the factor $\gamma(n)$ given by [2] is rather large.[7] Following Ajtai's seminal work, Cai and Nerurkar [11] showed that finding small solutions to Ajtai's random linear equations (with nonnegligible probability $\delta(n) = 1/n^{O(1)}$) is at least as hard as computing maximal sets of linearly independent vectors that are within a worst-case factor $n^{3+\epsilon}$ from the shortest[8] (for any fixed $\epsilon > 0$, independently of the success probability $\delta(n)$). It immediately follows (using standard relations between lattice problems [26, 7]) that Ajtai's random problem is at least as hard to solve on the average as approximating the length of the shortest nonzero vector in any $n$-dimensional lattice within a factor $\gamma(n) = n^{4+\epsilon}$ in the worst case.[9]

The question of determining under what conditions the number of lattice points inside a convex body $\mathcal{Q}$ is roughly proportional to the volume has been extensively studied, but mostly for the case of the integer lattice $\mathbb{Z}^n$. For example Mazo and Odlyzko [29] study the problem when $\mathcal{Q}$ is a sphere of radius $r$, in connection with

[5]To be precise, [2] proves only the result for $\delta(n) = 1/2$ and remarks that the proof can be generalized to any nonnegligible $\delta(n) = 1/n^{O(1)}$.

[6]It should be remarked that, as observed in [2], setting $\delta(n) = 1/2$ already gives weak one-way functions, which can be transformed (using standard techniques; see [16]) into strong one-way functions based on the worst-case hardness of approximating the shortest vector problem (SVP) within a fixed polynomial factor $\gamma(n) = n^{O(1)}$. However, in order to argue that no efficient algorithm can solve Ajtai's original problem with nonnegligible probability, [2] seems to require that no efficient algorithm can approximate the worst-case lattice problems within *any* polynomial factors.

[7]No specific value of $\gamma(n)$ is given in [2], but [11] estimates that a factor $\gamma(n) = n^8$ can be derived from the proof.

[8]Cai and Nerurkar [11] prove only the result for the shortest basis problem, but it is easy to modify their proof to yield a result for the shortest independent vector problem (SIVP). (See footnote 9 for more information.)

[9] To be precise, Cai and Nerurkar [11] claim only a $\gamma(n) = n^{5+\epsilon}$ connection factor, which is proved in three steps: (1) first they use small solutions (say, of size $O(n)$) of random linear equations to find linearly independent lattice vectors within an $n^{3+\epsilon}$ factor from the shortest *basis* in the lattice; (2) then, they use these linearly independent vectors to get an $n^{3.5+\epsilon}$ approximation to the shortest basis problem; and (3) finally, they connect the shortest basis problem to the SVP, losing an additional $n^{1.5}$ factor. We observe that the linear equations of [11] have solutions as small as $n^{0.5+\epsilon}$, which, if used in the proof, result in linearly independent lattice vectors within an $n^{2.5+\epsilon}$ factor from the shortest lattice basis. Then, we observe that the same technique used in [11] to transform these vectors into an $n^{3+\epsilon}$-approximate solution to the shortest basis problem can also be used to show that the original vectors are an $n^{3+\epsilon}$-approximate solution to the SIVP. This allows us to use known results [26, 7] to solve the SVP by losing only an additional factor $n$ (instead of $n^{1.5}$), leading to an $n^{4+\epsilon}$-approximate solution to the SVP.

universal quantization and low density subset-sum problems. (See [29] and references therein for a description of these problems.) In particular, they show that for $r = O(\sqrt{n})$ the number of integer lattice points in the sphere can deviate from the expected value by factors exponential in $n$, but they claim that if $r = n^{1/2+\epsilon}$ (for any $\epsilon > 0$), then the number of integer lattice points in the sphere is always asymptotic to the volume, no matter where the center is located. A different class of convex bodies is considered by Kannan and Vempala in [25], but, as usual, only for the special case of the integer lattice $\mathbb{Z}^n$. In [25] $\mathcal{Q}$ is an $n$-dimensional convex polytope with $m$ facets, and the result is that the number of integer lattice points in $\mathcal{Q}$ is approximately proportional to the volume provided that $\mathcal{Q}$ contains a sphere of radius $\Omega(n \cdot \sqrt{\log m})$.[10] A result for arbitrary convex bodies is proved by Dyer, Frieze, and Kannan [14], who show that the number of integer lattice points in $\mathcal{Q}$ is approximately proportional to the volume of $\mathcal{Q}$, provided $\mathcal{Q}$ contains a sphere of radius $\Omega(n^{1.5})$. In section 3 we generalize the result of [14] to arbitrary lattices and show that the number of lattice points in $\mathcal{Q}$ is approximately proportional to the volume provided that $\mathcal{Q}$ contains a sphere of radius $\Omega(n)$ times bigger than the covering radius of the lattice (see Theorem 3.6).

The CRP has been extensively studied from a mathematical point of view, leading, for example, to the transference theorems of Banaszczyk [7], but it has received little or no attention from a computational perspective. Two relevant results about the CRP are McLoughlin's proof [30] that the analogous problem on linear codes is hard for the second level of the polynomial hierarchy and Kannan's algorithm [24] showing that a variant of the CRP for lattices (where the norm defined by an input parallelotope is used, instead of the usual Euclidean norm) can be solved in polynomial time for any fixed dimension. Partly motivated by our work [35], some initial progress toward understanding the computational complexity of (approximately) computing the covering radius of lattices and linear codes has been recently made by Guruswami, Micciancio, and Regev [20]. The reader is referred to that paper and section 2.3 for further information about the computational complexity of the CRP.

The problem of decoding specific lattices has been considered in coding theory, for example, in connection with vector quantization. In [12] Conway and Sloane give polynomial time decoding algorithms for the root lattices $A_n, D_n$ and their duals $A_n^*, D_n^*$, as well as various other low-dimensional lattices.[11] From a computational complexity point of view, the problem has been considered under the name *closest vector problem with preprocessing*. Adapting similar results of Bruck and Naor [9] and Lobstein [28] for coding and subset-sum problems, Micciancio [31] showed that there are sequences of lattices such that solving the CVP is NP-hard. These results have been improved by Feige and Micciancio [15] and then Regev [38] to show that (unless P = NP) there are lattices and codes that cannot be efficiently decoded even approximately for any constant approximation factor smaller than $\sqrt{3}$. Notice that the goal of [31, 15, 38] is opposite ours: while [31, 15, 38] give explicit constructions of lattices that cannot be easily decoded, in this paper we search for explicit constructions of easily decodable lattices.

Almost perfect lattices have been extensively studied from a mathematical point of

[10] As a side remark, the motivation to study this problem in [25] is somehow opposite ours, as they count the number of lattice points in a polytope to estimate its volume. Here, we try to get a bound on the number of lattice points for convex bodies $\mathcal{Q}$ of known volume.

[11] Asymptotically, only results for infinite families of lattices are interesting because the CVP is known to be solvable in polynomial time in any fixed dimension [23].

view. In particular, Rogers [40] proved that there exist $\tau(n)$-perfect lattices for $\tau(n) < 3$ and any dimension $n$, and Butler [10] improved the result to $\tau(n) = 2 + o(1)$. Our exponential time construction of almost perfect lattices in Theorem 4.4 is essentially an algorithmic variant of Rogers's proof.

In this paper we consider the worst-case complexity of computing short vectors (as well as solving other computational lattice approximation problems) in *any* lattice. In a recent breakthrough paper [39], Regev has given encryption schemes and collision resistant hash functions that are as hard to break as computing shortest nonzero vectors in lattices with *special structure*. The results proved in [39] achieve approximation factors $O(n^{1.5})$ which are smaller than any other known reduction, but only for lattices where the shortest vector is *unique* is some technical sense. This special structure is common in the construction of lattice-based public key encryption schemes [4] but does not seem necessary to build one-way or collision resistant hash functions. In section 8 we explain how the techniques presented in this paper might lead to one-way and collision resistant hash functions that are as hard to break as solving the SVP (or other lattice problems) in *any* lattice, within approximation factors similar to those established in [39] for the special class of lattices possessing unique shortest vectors.

Another relevant paper establishing results similar to ours, but for a special class of lattices, is [34], where the goal is to obtain hard-on-average problems with very compact representation, rather than improving the connection factor. In [34] Micciancio shows that if approximating the SVP (or various other lattice problems) is hard in the worst case over a class of lattices with a special *cyclic* property, then one can define random linear equations with only $\omega(\log n)$ variables that are hard to solve on the average. This yields random equations (and cryptographic one-way functions) with a much smaller representation size than those considered in this paper, possibly leading to practical and provably secure lattice-based cryptographic functions.

**1.4. Outline.** The rest of the paper is organized as follows. In section 2 we introduce basic definitions and notation used throughout the paper and give background about lattice problems and computational complexity. In section 3 we define the (generalized) uniform radius and relate it to other lattice quantities. In section 4 we initiate the algorithmic study of almost perfect lattices and present the first polynomial time construction of easily decodable almost perfect lattices. These lattices are used in section 5 to define a new random class of equations that generalizes Ajtai's. In section 6 we prove the main technical result of the paper: finding small solutions to the random linear equations of section 5 is at least as hard as finding short (relative to the generalized uniform radius) linearly independent lattice vectors in the worst case. In section 7 we relate this problem to other well-known lattice problems, like approximating the length of the shortest vector in a lattice. Section 8 concludes with a brief summary of our main results and some open problems whose solution would allow us to further improve the connection factors established in this paper.

## 2. Preliminaries.

In this section, we introduce the notation that will be used in the rest of the paper, and then we briefly recall basic notions about lattices, statistical distance, and iterative reductions. For more background material about lattices the reader is referred to the book [36].

**2.1. Notation.** For any finite set $S$, the size of $S$ is denoted $\#S$. For any real $x$, $\lfloor x \rfloor$ denotes the largest integer not greater than $x$, and $\lfloor x \rceil = \lfloor x + 1/2 \rfloor$ is the rounding of $x$ to the closest integer. For any string $s$, the length of $s$ is denoted $|s|$. For any positive real $\epsilon > 0$, we write $[1 \pm \epsilon]$ to denote the interval $[1-\epsilon, 1+\epsilon]$. Arithmetic operations on intervals are defined in the obvious way by extending the standard arithmetic operations pointwise; e.g., $x \cdot [1 \pm \epsilon]$ denotes the interval $[x - \epsilon x, x + \epsilon x]$. Notice that $a \in b \cdot [1 \pm \epsilon]$ if and only if the relative additive error $|a-b|/|b|$ is at most $\epsilon$. For any two positive reals $a, b \geq 0$, we write $a \gtrsim b$ if $a \geq (1/2) \cdot b$ and $a \lesssim b$ if $a \leq (3/2) \cdot b$. We say that $a$ is approximately equal to $b$ (written $a \approx b$) if both $a \lesssim b$ and $a \gtrsim b$, i.e., $a \in b \cdot [1 \pm 1/2]$. Notice that $a \approx b$ is not a symmetric relation; i.e., $a \approx b$ does not imply $b \approx a$. For any $a, b, c \geq 0$, if $a \approx c$ and $b \approx c$, then $a$ and $b$ are within a factor 3 one from the other, i.e.,

$$\forall a, b, c.\ (a \approx c) \wedge (b \approx c) \Rightarrow (a/3 \leq b \leq 3a). \tag{2.1}$$

In the paper we use the standard asymptotic notation for functions. For any two positive real functions $f, g$, we write $f = O(g)$ or $g = \Omega(f)$ if there exists a constant $c > 0$ such that $f(x) \leq c \cdot g(x)$ for all sufficiently large $x$. We write $f = \Theta(g)$ if $f = O(g)$ and $f = \Omega(g)$. We write $f = o(g)$ or $g = \omega(f)$ if $f(x) < c \cdot g(x)$ for all $c > 0$ and all sufficiently large $x$. We also use notation $O(f)$, (resp., $\Omega(f), o(f), \omega(f)$) to denote the class of all functions $g$ such that $g = O(f)$ (resp., $g = \Omega(f)$, etc.), or an arbitrary, but fixed, function from that class. For example, we write $O(1)$ to denote an arbitrary constant or $n^{O(1)}$ to denote an arbitrary polynomially bounded function of $n$. A function $f$ is negligible if $f(n) = n^{-\omega(1)}$, i.e., if $f(n)$ is asymptotically smaller than any inverse polynomial in $n$.

Let $\mathbb{R}$, $\mathbb{Q}$, and $\mathbb{Z}$ be the sets of the reals, the rationals, and the integers, respectively. The $m$-dimensional Euclidean space is denoted $\mathbb{R}^m$. We use bold lowercase letters (e.g., $\mathbf{x}$) to denote vectors and bold uppercase letters (e.g., $\mathbf{M}$) to denote matrices. The $n$-dimensional identity matrix, i.e., the $n \times n$ diagonal matrix with 1's on the diagonal, is denoted $\mathbf{I}_n$ or simply $\mathbf{I}$ when the dimension is clear from the context. The columns of a matrix $\mathbf{M}$ are usually denoted by the corresponding lowercase letters, e.g., $\mathbf{m}_1, \ldots, \mathbf{m}_n$. As an exception, the standard unit vectors, i.e., the columns of the identity matrix $\mathbf{I}_n$, are denoted $\mathbf{e}_1, \ldots, \mathbf{e}_n$. If $\mathcal{Q} \subseteq \mathbb{R}^n$ is an arbitrary region of space and $\mathbf{x} \in \mathbb{R}^n$ is a vector, $\mathcal{Q} + \mathbf{x} = \{\mathbf{y} + \mathbf{x} : \mathbf{y} \in \mathcal{Q}\}$ denotes a copy of $\mathcal{Q}$ shifted by $\mathbf{x}$. The $\ell_2$ norm of a vector $\mathbf{x} \in \mathbb{R}^n$ is $\|\mathbf{x}\| = \sqrt{\sum x_i^2}$, and the associated distance is $\operatorname{dist}(\mathbf{x}, \mathbf{y}) = \|\mathbf{x} - \mathbf{y}\|$. For a matrix $\mathbf{M} = [\mathbf{m}_1, \ldots, \mathbf{m}_n]$, we define $\|\mathbf{M}\| = \max_i \|\mathbf{m}_i\|$, where $\mathbf{m}_i$ are the columns of $\mathbf{M}$. For vector $\mathbf{v} \in \mathbb{R}^n$ and set $\mathcal{Q} \subseteq \mathbb{R}^n$, let $\operatorname{dist}(\mathbf{v}, \mathcal{Q}) = \inf_{\mathbf{w} \in \mathcal{Q}} \|\mathbf{v} - \mathbf{w}\|$ be the distance between $\mathbf{v}$ and $\mathcal{Q}$. For vector $\mathbf{v} \in \mathbb{R}^n$ and real $r$, let $\mathcal{B}(\mathbf{v}, r) = \{\mathbf{w} \in \mathbb{R}^n : \operatorname{dist}(\mathbf{v}, \mathbf{w}) < r\}$ be the open ball of radius $r$ centered in $\mathbf{v}$ and $\bar{\mathcal{B}}(\mathbf{v}, r) = \{\mathbf{w} \in \mathbb{R}^n : \operatorname{dist}(\mathbf{v}, \mathbf{w}) \leq r\}$ its topological closure. When the center $\mathbf{v} = \mathbf{0}$ is the origin, we simply write $\mathcal{B}(r)$ and $\bar{\mathcal{B}}(r)$. We often use matrix notation to denote sets of vectors. For example, matrix $\mathbf{S} \in \mathbb{R}^{m \times n}$ represents the set of $m$-dimensional vectors $\{\mathbf{s}_1, \ldots, \mathbf{s}_n\}$, where $\mathbf{s}_1, \ldots, \mathbf{s}_n$ are the columns of $\mathbf{S}$. The linear space spanned by a set of vectors $\mathbf{S}$ is denoted $\operatorname{span}(\mathbf{S}) = \{\sum_i x_i \cdot \mathbf{s}_i : \text{for all } i.x_i \in \mathbb{R}\}$. For any set of linearly independent vectors $\mathbf{S}$, we define the centered half-open parallelepiped $\mathcal{P}(\mathbf{S}) = \{\mathbf{S}\mathbf{x} : -1/2 \leq x_i < 1/2\}$.

**2.2. Lattices.** An $m$-dimensional *lattice* is the set of all integer combinations $\{\sum_{i=1}^n x_i \mathbf{b}_i : x_i \in \mathbb{Z}\}$ of $n$ linearly independent vectors $\mathbf{b}_1, \ldots, \mathbf{b}_n$ in $\mathbb{R}^m$ ($m \geq n$). The set of vectors $\mathbf{b}_1, \ldots, \mathbf{b}_n$ is called a *basis* for the lattice, and the integer $n = \dim(\operatorname{span}(\mathbf{B}))$ is called the *rank* of the lattice. If the rank $n$ equals the dimension $m$, then the lattice is called *full rank* or *full dimensional*. Lattices are infinite Abelian groups with respect to the vector addition operation and can be equivalently defined as discrete additive subgroups of $\mathbb{R}^m$. A basis can be compactly represented by the matrix $\mathbf{B} = [\mathbf{b}_1|\dots|\mathbf{b}_n] \in \mathbb{R}^{m\times n}$ having the basis vectors as columns. The lattice generated by $\mathbf{B}$ is denoted $\mathcal{L}(\mathbf{B})$. Notice that $\mathcal{L}(\mathbf{B}) = \{\mathbf{Bx}\colon \mathbf{x} \in \mathbb{Z}^n\}$, where $\mathbf{Bx}$ is the usual matrix-vector multiplication. We use the notation $\mathcal{L}(\mathbf{B})$ to denote the set $\{\mathbf{Bx}\colon \mathbf{x} \in \mathbb{Z}^n\}$ even when vectors $\mathbf{B}$ are not linearly independent. The dual of a lattice $\mathcal{L}(\mathbf{B})$ is the set

$$\mathcal{L}(\mathbf{B})^* = \{\mathbf{x} \in \operatorname{span}(\mathcal{L}(\mathbf{B}))\colon \forall \mathbf{y} \in \mathcal{L}(\mathbf{B}).\langle \mathbf{x}, \mathbf{y}\rangle \in \mathbb{Z}\}$$

of all vectors in the linear span of $\mathcal{L}(\mathbf{B})$ that have integer scalar product with all lattice vectors. The dual of a lattice is a lattice, and a possible basis for the dual of $\mathcal{L}(\mathbf{B})$ is given by $\mathbf{B}^* = \mathbf{B}(\mathbf{B}^T\mathbf{B})^{-1}$, where $\mathbf{B}^T$ is the transpose of $\mathbf{B}$. (Notice that if $\mathbf{B}$ is a basis, it has full column rank and the square matrix $\mathbf{B}^T\mathbf{B}$ is invertible.)

The *minimum distance* of a lattice $\mathcal{L}(\mathbf{B})$ (denoted $\lambda_1(\mathcal{L}(\mathbf{B}))$) is the minimum distance between any two (distinct) lattice points and equals the length of the shortest nonzero lattice vector:

$$\lambda_1(\mathcal{L}(\mathbf{B})) = \min\{\operatorname{dist}(\mathbf{x}, \mathbf{y}) : \mathbf{x} \neq \mathbf{y} \in \mathcal{L}(\mathbf{B})\} = \min\{\|\mathbf{x}\| : \mathbf{x} \in \mathcal{L}(\mathbf{B}) \setminus \{\mathbf{0}\}\}.$$

This definition can be generalized to define the $i$th successive minimum as the smallest $\lambda_i$ such that $\bar{\mathcal{B}}(\lambda_i)$ contains $i$ linearly independent lattice points:

$$\lambda_i(\mathcal{L}(\mathbf{B})) = \min\{r\colon \dim(\operatorname{span}(\mathcal{L}(\mathbf{B}) \cap \bar{\mathcal{B}}(r))) \geq i\}.$$

Another important constant associated to a lattice is the covering radius. The covering radius $\rho(\mathcal{L}(\mathbf{B}))$ of a lattice is the maximum distance $\operatorname{dist}(\mathbf{x}, \mathcal{L}(\mathbf{B}))$ when $\mathbf{x}$ ranges over the linear span of $\mathbf{B}$:

$$\rho(\mathcal{L}(\mathbf{B})) = \max_{\mathbf{x}\in\operatorname{span}(\mathbf{B})}\{\operatorname{dist}(\mathbf{x}, \mathcal{L}(\mathbf{B}))\}.$$

A sublattice of $\mathcal{L}(\mathbf{B})$ is a lattice $\mathcal{L}(\mathbf{S})$ such that $\mathcal{L}(\mathbf{S}) \subseteq \mathcal{L}(\mathbf{B})$. $\mathcal{L}(\mathbf{S})$ is a *full rank* sublattice of $\mathcal{L}(\mathbf{B})$ if it has the same rank as $\mathcal{L}(\mathbf{B})$. The determinant of a (rank $n$) lattice $\det(\mathcal{L}(\mathbf{B}))$ is the ($n$-dimensional) volume of the fundamental parallelepiped $\mathcal{P}(\mathbf{B})$, and it does not depend on the choice of the basis $\mathbf{B}$. If $\mathcal{L}(\mathbf{B})$ is full dimensional, then $\det(\mathcal{L}(\mathbf{B}))$ equals the absolute value of the determinant of the $n \times n$ basis matrix $|\det(\mathbf{B})|$. Hadamard's bound gives a simple way to bound the determinant of a lattice as $\det(\mathcal{L}(\mathbf{B})) \leq \prod_i \|\mathbf{b}_i\|$. Hadamard's bound can be much larger than the actual value of the determinant, and it equals the determinant if and only if the basis $\mathbf{B}$ is orthogonal. Minkowski's first theorem (see [36, pp. 11–14]) implies that any rank $n$ lattice $\mathcal{L}(\mathbf{B})$ contains a nonzero vector of length at most

$$\lambda_1(\mathcal{L}(\mathbf{B})) \leq \sqrt{n}\det(\mathcal{L}(\mathbf{B}))^{1/n}. \tag{2.2}$$

The Voronoi cells of a lattice, defined below, play an important role in our proofs. For uniformity, and by analogy with the definition of the half-open parallelepiped, we first define the half-open Voronoi cells. However, we remark that in the rest of the paper we need only the more standard notions of open and closed Voronoi cells.

Definition 2.1. *Let $\preceq$ be the total order on $\mathbb{R}^n$ where $\mathbf{x} \preceq \mathbf{y}$ if and only if $\|\mathbf{x}\| < \|\mathbf{y}\|$ or $\|\mathbf{x}\| = \|\mathbf{y}\|$ and $\mathbf{x}$ precedes $\mathbf{y}$ lexicographically; i.e., $x_i < y_i$ for the first coordinate $i$ such that $x_i \neq y_i$. For any lattice $\mathcal{L}(\mathbf{B})$ and lattice point $\mathbf{x} \in \mathcal{L}(\mathbf{B})$, the (half-open)* Voronoi *cell of $\mathbf{x}$ is the set*

$$\mathcal{V}(\mathbf{x}, \mathcal{L}(\mathbf{B})) = \{\mathbf{z} \in \operatorname{span}(\mathcal{L}(\mathbf{B}))\colon \forall \mathbf{y} \in \mathcal{L}(\mathbf{B}).(\mathbf{z} - \mathbf{x}) \preceq (\mathbf{z} - \mathbf{y})\}.$$

*The closed (resp., open) Voronoi cell* $\bar{\mathcal{V}}(\mathbf{x}, \mathcal{L}(\mathbf{B}))$ *(resp.,* $\mathcal{V}^o(\mathbf{x}, \mathcal{L}(\mathbf{B}))$*) is defined as the topological closure (resp., interior) of* $\mathcal{V}(\mathbf{x}, \mathcal{L}(\mathbf{B}))$.

For simplicity, the Voronoi cell of the origin $\mathbf{x} = \mathbf{0}$ is denoted $\mathcal{V}(\mathcal{L}(\mathbf{B}))$. Notice that the Voronoi cell of the integer lattice equals the half-open unit cube: $\mathcal{V}(\mathbb{Z}^n) = \mathcal{P}(\mathbf{I}_n)$. We need some simple properties about Voronoi cells, as listed below. All properties are easily verified and their proof is left to the reader.

Proposition 2.2. *For any lattice* $\mathcal{L}(\mathbf{B})$*, the Voronoi cells of* $\mathcal{L}(\mathbf{B})$ *satisfy the following properties:*

(i) *The half-open Voronoi cells form a partition of* $\operatorname{span}(\mathcal{L}(\mathbf{B}))$*; i.e., for any* $\mathbf{y} \in \operatorname{span}(\mathcal{L}(\mathbf{B}))$ *there exists a unique lattice point* $\mathbf{x} \in \mathcal{L}(\mathbf{B})$ *such that* $\mathbf{y} \in \mathcal{V}(\mathbf{x}, \mathcal{L}(\mathbf{B}))$.

(ii) *All Voronoi cells* $\mathcal{V}(\mathbf{x}, \mathcal{L}(\mathbf{B}))$ *(for* $\mathbf{x} \in \mathcal{L}(\mathbf{B})$*) are shifted copies* $\mathcal{V}(\mathbf{x}, \mathcal{L}(\mathbf{B})) = \mathcal{V}(\mathcal{L}(\mathbf{B})) + \mathbf{x}$ *of the fundamental cell associated to the origin.*

(iii) *Each cell* $\mathcal{V}(\mathbf{x}, \mathcal{L}(\mathbf{B}))$ *contains the open sphere* $\mathcal{B}(\mathbf{x}, \lambda_1(\mathcal{L}(\mathbf{B}))/2)$*, and it is contained in the closed sphere* $\bar{\mathcal{B}}(\mathbf{x}, \rho(\mathcal{L}(\mathbf{B})))$.

(iv) *The volume of a Voronoi cell equals* $\operatorname{vol}(\mathcal{V}(\mathbf{x}, \mathcal{L}(\mathbf{B}))) = \det(\mathcal{L}(\mathbf{B}))$.

(v) *The open Voronoi cell* $\mathcal{V}^o(\mathbf{x}, \mathcal{L}(\mathbf{B}))$ *is the set of all points in* $\operatorname{span}(\mathcal{L}(\mathbf{B}))$ *which are strictly closer to* $\mathbf{x}$ *than to any other lattice point.*

(vi) *The closed Voronoi cell* $\bar{\mathcal{V}}(\mathbf{x}, \mathcal{L}(\mathbf{B}))$ *is the set of all points in* $\operatorname{span}(\mathcal{L}(\mathbf{B}))$ *which are at least as close to* $\mathbf{x}$ *as to any other lattice point.*

(vii) *The cells* $\mathcal{V}^o(\mathbf{x}, \mathcal{L}(\mathbf{B}))$ *and* $\bar{\mathcal{V}}(\mathbf{x}, \mathcal{L}(\mathbf{B}))$ *are convex and symmetric about their center* $\mathbf{x}$.

**2.3. Computational problems on lattices.** When discussing computational issues related to lattices, it is customary to assume that the lattices are represented by a basis matrix $\mathbf{B}$ and that $\mathbf{B}$ has integer entries. Other representations are possible; e.g., a sublattice of $\mathbb{Z}^n$ can be defined as the set of integer solutions to a system of homogeneous modular linear equations. These alternative representations are computationally equivalent to giving a basis, i.e., for example, given a system of homogeneous modular linear equations one can compute in polynomial time a basis for the corresponding lattice.

In this paper we consider the following problems on lattices. All problems are defined in their approximation version, where the approximation factor $\gamma(n)$ can be a function of the rank $n$ of the lattice. The exact version of the problems corresponds to approximation factor $\gamma(n) = 1$.

Definition 2.3. *The* shortest vector problem (SVP)*, given a lattice basis* $\mathbf{B}$*, asks for a nonzero lattice vector* $\mathbf{v} \in \mathcal{L}(\mathbf{B})$ *of length at most* $\gamma(n) \cdot \lambda_1(\mathcal{L}(\mathbf{B}))$*, where* $n$ *is the rank of* $\mathbf{B}$ *and* $\gamma(n) \geq 1$ *is the approximation factor. The problem can be defined also in a* length estimation *version, where given a basis* $\mathbf{B}$*, one has only to find a value* $\hat{\lambda}_1$ *such that* $\lambda_1(\mathcal{L}(\mathbf{B})) \leq \hat{\lambda}_1 \leq \gamma(n) \cdot \lambda_1(\mathcal{L}(\mathbf{B}))$*. The promise problem*[12] *naturally associated to the length estimation version of* SVP *(denoted* GapSVP$_\gamma$*) is as follows: given* $(\mathbf{B}, d)$*, where* $\mathbf{B}$ *is a lattice basis and* $d$ *is a (rational) number, decide if* $\lambda_1(\mathcal{L}(\mathbf{B})) \leq d$ *or* $\lambda_1(\mathcal{L}(\mathbf{B})) > \gamma(n) \cdot d$.

The promise problem is easily shown to be equivalent to the length estimation version of SVP. (See, for example, [36, pp. 20–21].) However, the promise and length

[12]Promise problems are a natural generalization of decision problems where one is asked whether a given input satisfies one of two mutually exclusive properties (e.g., tell if a given input lattice contains a short nonzero vector, or it does not). However, differently from decision problems, the two properties are not necessarily exhaustive. The problem is, under the promise that the given input satisfies one of the two conditions, tell which of the two properties is satisfied. If the input satisfies neither property, then any answer is acceptable.

estimation problems are not known to be equivalent to the search (vector finding) version of SVP for $\gamma(n) > 1$; i.e., given an oracle to (approximately) compute the length of the shortest nonzero vector in any lattice, it is not clear how to find short lattice vectors.[13] The SVP is NP-hard (under randomized reductions), even in its promise version, for any constant approximation factor $\gamma(n) < \sqrt{2}$ [3, 33]. The promise version of the problem is clearly solvable in NP. For $\gamma(n) = \Omega(\sqrt{n/\log n})$, the problem is in coAM [17], and for $\gamma(n) = \Omega(\sqrt{n})$ it is also in coNP [1]. (See [26, 7] for earlier results with approximation factor $\gamma(n) = \Omega(n)$.) Finally, when $\gamma(n) = e^{\Omega(n \log\log n/\log n)}$ the problem can be solved in random polynomial time [5], and deterministic polynomial time solutions are known only for $\gamma(n) = e^{\Omega(n(\log\log n)^2/\log n)}$ [41].

DEFINITION 2.4. *The* shortest independent vectors problem (SIVP)*, given a lattice basis* $\mathbf{B}$ *of rank* $n$*, asks for a set of* $n$ *linearly independent lattice vectors* $\mathbf{S} \subseteq \mathcal{L}(\mathbf{B})$ *such that* $\|\mathbf{S}\| \leq \gamma(n) \cdot \lambda_n(\mathcal{L}(\mathbf{B}))$*. The problem can be defined also in a* length estimation *version, where given a basis* $\mathbf{B}$*, one has only to find a value* $\hat{\lambda}_n$ *such that* $\lambda_n(\mathcal{L}(\mathbf{B})) \leq \hat{\lambda}_n \leq \gamma(n) \cdot \lambda_n(\mathcal{L}(\mathbf{B}))$*. The promise problem naturally associated to the length estimation version of* SIVP *(denoted* GAPSIVP$_\gamma$*) is as follows: given* $(\mathbf{B}, d)$*, where* $\mathbf{B}$ *is a lattice basis and* $d$ *is a (rational) number, decide if* $\lambda_n(\mathcal{L}(\mathbf{B})) \leq d$ *or* $\lambda_n(\mathcal{L}(\mathbf{B})) > \gamma(n) \cdot d$*.*

SIVP is NP-hard (as usual, already in its promise version) for any constant approximation factor [8]. The (promise version of) SIVP is clearly in NP. For $\gamma(n) = \Omega(\sqrt{n/\log n})$ the problem is in coAM [20, 17], and for $\gamma(n) = \Omega(\sqrt{n})$ it is also in coNP [20, 1]. On the algorithmic side, it is possible to reduce approximating SIVP within a factor $\sqrt{n}\cdot\gamma(n)$ to approximating SVP within a factor $\gamma(n)$, where both SIVP and SVP are considered in their search version. (See, for example, [36, Chapter 7].) For the promise version of the problems, the transference theorems of [26, 7] immediately give a reduction from GAPSIVP$_{n\cdot\gamma(n)}$ to (the complement of) GAPSVP$_{\gamma(n)}$. These reductions from SIVP to SVP immediately give deterministic polynomial time algorithms for approximating SIVP within factors $\gamma(n) = e^{O(n(\log\log n)^2/\log n)}$ and probabilistic polynomial time algorithms for $\gamma(n) = e^{O(n(\log\log n)/\log n)}$.

DEFINITION 2.5. *The* covering radius problem (CRP)*, given a lattice basis* $\mathbf{B}$*, asks for a value* $\hat{\rho}$ *such that* $\rho(\mathcal{L}(\mathbf{B})) \leq \hat{\rho} \leq \gamma(n) \cdot \rho(\mathcal{L}(\mathbf{B}))$*. The promise problem naturally associated to* CRP*, (denoted* GAPCRP$_\gamma$*) is as follows: given* $(\mathbf{B}, d)$ *where* $\mathbf{B}$ *is a lattice basis and* $d$ *is a (rational) number, decide if* $\rho(\mathcal{L}(\mathbf{B})) \leq d$ *or* $\rho(\mathcal{L}(\mathbf{B})) > \gamma(n) \cdot d$*.*

Currently, no NP-hardness result is known for CRP. However, we do not even know how to solve the problem (in its exact version, i.e., for $\gamma(n) = 1$) in nondeterministic polynomial time (NP), and the analogous problem for linear codes is known to be hard for the second level of the polynomial hierarchy [30]. So, we can reasonably conjecture that the same is true for the CRP on lattices.

*Conjecture.* The CRP for lattices (GAPCRP$_1$) is $\Pi_2$-hard.

Recently, [20] has shown that GAPCRP$_\gamma$ is in AM for $\gamma = 2$, in coAM for $\gamma(n) = \Omega(\sqrt{n/\log(n)})$, and in NP $\cap$ coNP for $\gamma(n) = \Omega(\sqrt{n})$. The problem can also be approximated within $\gamma = 1 + \epsilon$ (for any constant $\epsilon > 0$) in random exponential time [20], $\gamma(n) = e^{O(n(\log\log n)/\log n)}$ in random polynomial time, and $\gamma(n) = e^{O(n(\log\log n)^2/\log n)}$ in deterministic polynomial time.

[13] A reduction for the exact case ($\gamma = 1$) is given in [22]. This is the only direct reduction known to date. Technically, a (trivial) reduction between the two problems also exists for approximation factors $\gamma$ for which approximating $\lambda_1$ is NP-hard or finding short vectors is solvable in polynomial time. No reduction is known for any other intermediate approximation factor.

DEFINITION 2.6. *The* closest vector problem (CVP)*, given a lattice basis* $\mathbf{B}$ *and target vector* $\mathbf{t}$*, asks for a lattice point* $\mathbf{v} \in \mathcal{L}(\mathbf{B})$ *such that* $\operatorname{dist}(\mathbf{t}, \mathbf{v}) \leq \gamma(n) \cdot \operatorname{dist}(\mathbf{t}, \mathcal{L}(\mathbf{B}))$*. The problem can be defined also in a* distance estimation *version, where given a basis* $\mathbf{B}$ *and target* $\mathbf{t}$*, one has only to find a value* $\hat{d}$ *such that* $\operatorname{dist}(\mathbf{t}, \mathcal{L}(\mathbf{B})) \leq \hat{d} \leq \gamma(n) \cdot \operatorname{dist}(\mathbf{t}, \mathcal{L}(\mathbf{B}))$*. The promise problem naturally associated to* CVP *(denoted* GAPCVP$_\gamma$*) is as follows: given* $(\mathbf{B}, \mathbf{t}, d)$*, where* $\mathbf{B}$ *is a lattice basis,* $\mathbf{t}$ *is a target vector, and* $d$ *is a (rational) number, decide if* $\operatorname{dist}(\mathbf{t}, \mathcal{L}(\mathbf{B})) \leq d$ *or* $\operatorname{dist}(\mathbf{t}, \mathcal{L}(\mathbf{B})) > \gamma(n) \cdot d$*.*

The CVP is known to be at least as hard as the SVP [19] for any approximation factor $\gamma(n)$. Moreover, it is NP-hard for quasi-polynomial approximation factors $\gamma(n) = n^{1/O(\log\log n)}$ [13]. For $\gamma(n) = \Omega(\sqrt{n/\log n})$ the problem is in coAM [17], and for $\gamma(n) = \Omega(\sqrt{n})$ it is also in coNP [1]. (See [26, 7, 38] for earlier results with approximation factor $\gamma(n) = \Omega(n)$.) Finally, the problem can be approximated in deterministic polynomial time within $\gamma(n) = e^{\Omega(n(\log\log n)^2/\log n)}$ [41, 22].

In the CVP, the target point $\mathbf{t}$ can be arbitrarily far from the lattice. In coding theory, Vardy [42] has considered a variant of the CVP where the distance of the target from the code is guaranteed to be less than the packing radius of the code. This problem (called the *bounded distance decoding* problem (BDD)) is interesting because decoding within the packing radius, if solvable, has a unique solution. (For this reason, the packing radius is sometimes also called the "unique decoding" radius.) For lattices, the analogous problem would be the following: given a lattice $\mathbf{B}$ and a point $\mathbf{t}$ within distance $d < \lambda_1(\mathcal{L}(\mathbf{B}))/2$ from $\mathcal{L}(\mathbf{B})$, find the (unique) lattice point within distance $d$ from $\mathbf{t}$. In general we can consider a similar problem for values of $d$ different from $\lambda_1(\mathcal{L}(\mathbf{B}))/2$, although when $d \geq \lambda_1(\mathcal{L}(\mathbf{B}))/2$ the solution is not necessarily unique. We consider the case when $d = \rho(\mathcal{L}(\mathbf{B}))$ equals the covering radius of the lattice. This case is interesting because there is always a lattice point within distance $\rho(\mathcal{L}(\mathbf{B}))$ from the target. Below we formally define an approximation version of this problem. Since for any lattice $\mathcal{L}(\mathbf{B})$ and target $\mathbf{t}$, there is always a lattice point within distance $\rho(\mathcal{L}(\mathbf{B}))$ from $\mathbf{t}$, we do not define distance estimation or promise versions of this problem.

DEFINITION 2.7. *The* guaranteed distance decoding *problem* (GDD$_\gamma$)*, given a lattice* $\mathbf{B}$ *and a target point* $\mathbf{t} \in \operatorname{span}(\mathbf{B})$*, asks for a lattice point* $\mathbf{x} \in \mathcal{L}(\mathbf{B})$ *such that* $\operatorname{dist}(\mathbf{t}, \mathbf{x}) \leq \gamma(n)\rho(\mathcal{L}(\mathbf{B}))$*, where* $n$ *is the rank of the lattice.*

The following relations are known among the parameters of a lattice $\mathcal{L}(\mathbf{B})$.

PROPOSITION 2.8. *For any rank* $n$ *lattice* $\mathbf{B}$*,*

$$\lambda_1(\mathcal{L}(\mathbf{B})) \leq \lambda_n(\mathcal{L}(\mathbf{B})) \leq 2\rho(\mathcal{L}(\mathbf{B})) \leq \sqrt{n}\lambda_n(\mathcal{L}(\mathbf{B})). \tag{2.3}$$

*Moreover, if* $\mathcal{L}(\mathbf{B})^*$ *is the dual lattice of* $\mathcal{L}(\mathbf{B})$*, then*

$$1 \leq \lambda_1(\mathcal{L}(\mathbf{B}))2\rho(\mathcal{L}(\mathbf{B})^*) \leq n \tag{2.4}$$

*and*

$$1 \leq \lambda_1(\mathcal{L}(\mathbf{B}))\lambda_n(\mathcal{L}(\mathbf{B})^*) \leq n. \tag{2.5}$$

*Proof.* See [36, Theorem 7.9] for (2.3) and [7] for (2.4) and (2.5). ◻

**2.4. Lattices and groups.** Let $\mathcal{L}(\mathbf{L})$ be a lattice. Any sublattice $\mathcal{L}(\mathbf{M}) \subseteq \mathcal{L}(\mathbf{L})$ defines a natural equivalence relation on $\mathcal{L}(\mathbf{L})$ as follows: two lattice points $\mathbf{x}, \mathbf{y} \in \mathcal{L}(\mathbf{L})$ are equivalent (written $\mathbf{x} \equiv_{\mathbf{M}} \mathbf{y}$) if and only if $\mathbf{x} - \mathbf{y} \in \mathcal{L}(\mathbf{M})$. The reader can easily check that $\equiv_{\mathbf{M}}$ is an equivalence relation; i.e., it is reflexive ($\mathbf{x} \equiv_{\mathbf{M}} \mathbf{x}$), symmetric ($\mathbf{x} \equiv_{\mathbf{M}} \mathbf{y} \Leftrightarrow \mathbf{y} \equiv_{\mathbf{M}} \mathbf{x}$), and transitive ($\mathbf{x} \equiv_{\mathbf{M}} \mathbf{y} \wedge \mathbf{y} \equiv_{\mathbf{M}} \mathbf{z} \Rightarrow \mathbf{x} \equiv_{\mathbf{M}} \mathbf{z}$). The $\equiv_{\mathbf{M}}$-equivalence class of $\mathbf{x} \in \mathcal{L}(\mathbf{L})$ (denoted $[\mathbf{x}]_{\mathbf{M}}$) is the set of all $\mathbf{y} \in \mathcal{L}(\mathbf{L})$ such that $\mathbf{x} \equiv_{\mathbf{M}} \mathbf{y}$. The quotient $\mathcal{L}(\mathbf{L})/\mathcal{L}(\mathbf{M}) = \{[\mathbf{x}]_{\mathbf{M}}: \mathbf{x} \in \mathcal{L}(\mathbf{L})\}$ is the set of all $\equiv_{\mathbf{M}}$-equivalence classes of $\mathcal{L}(\mathbf{L})$. The equivalence relation $\equiv_{\mathbf{M}}$ is a congruence with respect to the addition operation; i.e., if $\mathbf{x} \equiv_{\mathbf{M}} \mathbf{x}'$ and $\mathbf{y} \equiv_{\mathbf{M}} \mathbf{y}'$, then $(\mathbf{x}+\mathbf{y}) \equiv_{\mathbf{M}} (\mathbf{x}'+\mathbf{y}')$. It follows that for any two equivalence classes $[\mathbf{x}]_{\mathbf{M}}$ and $[\mathbf{y}]_{\mathbf{M}}$, the sum $[\mathbf{x}]_{\mathbf{M}} + [\mathbf{y}]_{\mathbf{M}} = [\mathbf{x}+\mathbf{y}]_{\mathbf{M}}$ is well defined; i.e., it does not depend on the choice of representatives $\mathbf{x}$, $\mathbf{y}$, and the quotient $\mathcal{L}(\mathbf{L})/\mathcal{L}(\mathbf{M})$ is an additive group with the sum operation just described. Notice that if $\mathcal{L}(\mathbf{L})$ is regarded as an Abelian group, then sublattice $\mathcal{L}(\mathbf{M})$ is a subgroup of $\mathcal{L}(\mathbf{L})$ and $(\mathcal{L}(\mathbf{L})/\mathcal{L}(\mathbf{M}), +)$ is just the standard quotient group.

Group $\mathcal{L}(\mathbf{L})/\mathcal{L}(\mathbf{M})$ is finite if and only if $\mathcal{L}(\mathbf{M})$ is a full rank sublattice of $\mathcal{L}(\mathbf{L})$, in which case the cardinality of the group is

$$\#(\mathcal{L}(\mathbf{L})/\mathcal{L}(\mathbf{M})) = \frac{\det(\mathcal{L}(\mathbf{M}))}{\det(\mathcal{L}(\mathbf{L}))}.$$

Elements of this group can be represented using several standard techniques, e.g., selecting a unique representative from each equivalence class. It is easy to see that for every equivalence class $[\mathbf{x}]_{\mathbf{M}}$ there exists a unique element $\mathbf{x}' \in \mathcal{L}(\mathbf{L}) \cap \mathcal{P}(\mathbf{M})$ such that $\mathbf{x} \equiv_{\mathbf{M}} \mathbf{x}'$. So, a possible set of (unique) representatives is given by the set

$$\mathcal{L}(\mathbf{L}) \cap \mathcal{P}(\mathbf{M})$$

of all lattice points that belong to the half-open parallelepiped $\mathcal{P}(\mathbf{M})$. Given an arbitrary lattice point $\mathbf{x} \in \mathcal{L}(\mathbf{B})$, the corresponding representative can be efficiently computed as follows: write $\mathbf{x}$ as $\mathbf{Mz}$, let $z_i' = \lfloor z_i \rceil$ for all $i = 1, \dots, n$, and set $\mathbf{x}' = \mathbf{M}(\mathbf{z} - \mathbf{z}')$.

The representation of group elements using vectors in $\mathcal{P}(\mathbf{M}) \cap \mathcal{L}(\mathbf{L})$, although polynomial, is not very efficient. In particular, the number of bits necessary to store a single group element can be much larger than $\log_2 \#G$. Other more efficient ways to represent group elements are possible—for example, using the Hermite normal form or the Smith normal form. These representations allow us to store group elements using only $\log |G|$ bits and perform the group operations in linear time. The techniques described in this paper are largely independent from the way group elements are represented, so we do not elaborate on this any further, and we refer the reader to [32, 36] for more details.

Later in this paper we need to sample elements from group $G = \mathcal{L}(\mathbf{L})/\mathcal{L}(\mathbf{M})$ uniformly at random. This can be easily done using an elementary group theoretic technique described in the following proposition.

PROPOSITION 2.9. *Let $G$ be a finite Abelian group and $g_1, \dots, g_n$ a generating set for $G$. Then, if $d_1, \dots, d_n$ are chosen uniformly at random in $\{1, \dots, \#G\}$, then the group element*

$$g = \sum_{i=1}^{n} d_i g_i$$

*is distributed uniformly at random over $G$.*

*Proof.* Since elements $g_1, \dots, g_n$ generate the entire group, we know that for any group element $a \in G$ there exists an integer vector $\mathbf{d}_a = [d_{a,1}, \dots, d_{a,n}]$ such that $\sum_{i=1}^{n} d_{a,i} g_i = a$. Let $K$ be the set of all integer vectors $\mathbf{d} = [d_1, \dots, d_n]$ such that $\sum d_i g_i = 0$ (in $G$). Notice that for any $a \in G$ and $\mathbf{d} \in \mathbb{Z}^n$, $\sum_i d_i g_i = a$ if and only if $\mathbf{d} \in K + \mathbf{d}_a$. Therefore, if $d_1, \ldots, d_n$ are chosen uniformly at random in $\{1, \ldots, \#G\}$, then the probability that $\sum_i d_i g_i = a$ equals exactly the size of $(K + \mathbf{d}_a) \cap \{1, \ldots, \#G\}^n$ divided by $(\#G)^n$. Since $K$ is periodic modulo $\#G$ (i.e., it is invariant under translations by vectors in $\#G \cdot \mathbb{Z}^n$), all sets $(K + \mathbf{d}_a) \cap \{1, \ldots, \#G\}^n$ have the same size, and the probability that $\sum_i d_i g_i = a$ is the same for all $a \in G$. $\square$

Of particular interest in this paper are quotient groups $G = \mathcal{L}(\mathbf{L})/\mathcal{L}(\mathbf{M})$, where $\mathbf{M}$ defines an almost *orthogonal* sublattice of $\mathcal{L}(\mathbf{L})$. The following lemma gives a possible way to build almost orthogonal sublattices for any input lattice $\mathcal{L}(\mathbf{L})$.

LEMMA 2.10. *Let $\mathcal{L}(\mathbf{B})$ be a lattice of rank $n$, $\sigma$ be a positive real, and $D$ be a decoding procedure that on input a vector $\mathbf{y} \in \operatorname{span}(\mathbf{B})$ returns a lattice point $D(\mathbf{y}) \in \mathcal{L}(\mathbf{B})$ such that $\operatorname{dist}(D(\mathbf{y}), \mathbf{y}) \le \sigma$. For any $\alpha \ge 2\sqrt{n} \cdot \sigma$, one can efficiently find (with $n$ calls to $D$) a basis of a full rank sublattice $\mathbf{S} \subset \mathcal{L}(\mathbf{B})$ such that for all $\mathbf{x} \in \mathbb{R}^n$*

$$\|\mathbf{S}\mathbf{x}\| \approx \alpha \cdot \|\mathbf{x}\|.$$

*Proof.* Let $\mathbf{s}_i = D(\alpha \cdot \mathbf{t}_i)$, where $\mathbf{t}_1, \ldots, \mathbf{t}_n$ are an orthonormal basis of $\operatorname{span}(\mathbf{B})$; e.g., if $\mathcal{L}(\mathbf{B})$ is a full rank lattice, set $\mathbf{t}_i = \mathbf{e}_i$. Clearly $\mathbf{s}_i \in \mathcal{L}(\mathbf{B})$ for all $i = 1, \ldots, n$. Let $\mathbf{x} \in \mathbb{R}^n$ be an arbitrary vector. We want to prove that $\|\mathbf{S}\mathbf{x}\| \approx \alpha \cdot \|\mathbf{x}\|$. We know that $\mathbf{s}_i = \alpha \cdot \mathbf{t}_i + \mathbf{r}_i$, where $\|\mathbf{r}_i\| = \|D(\alpha \cdot \mathbf{t}_i) - \alpha \cdot \mathbf{t}_i\| \le \sigma$. Therefore,

$$\|\mathbf{S}\mathbf{x}\| = \|(\alpha \cdot \mathbf{T} + \mathbf{R})\mathbf{x}\| = \|\alpha \cdot \mathbf{T}\mathbf{x} + \mathbf{R}\mathbf{x}\|.$$

By the triangle inequality, and using $\|\mathbf{T}\mathbf{x}\| = \|\mathbf{x}\|$ (which follows from the fact that $\mathbf{T}$ is an orthonormal set of vectors), we get

$$\alpha \cdot \|\mathbf{x}\| - \|\mathbf{R}\mathbf{x}\| \le \|\mathbf{S}\mathbf{x}\| \le \alpha \cdot \|\mathbf{x}\| + \|\mathbf{R}\mathbf{x}\|.$$

So, we need to prove that $\|\mathbf{R}\mathbf{x}\| \le \frac{\alpha}{2}\|\mathbf{x}\|$. By the triangle inequality and Cauchy–Schwarz,

$$\|\mathbf{R}\mathbf{x}\| \le \sum_{i=1}^{n} \|\mathbf{r}_i\| \cdot |x_i| \le \sigma \cdot \sum_{i=1}^{n} |x_i| \le \sqrt{n}\sigma \cdot \|\mathbf{x}\| \le \frac{\alpha}{2} \cdot \|\mathbf{x}\|.$$

This proves that $\|\mathbf{S}\mathbf{x}\| \approx \alpha \cdot \|\mathbf{x}\|$. The linear independence of vectors $\mathbf{S}$ immediately follows because if $\mathbf{S}$ were linearly dependent, then one could find a nonzero vector $\mathbf{x}$ such that $\mathbf{S}\mathbf{x} = \mathbf{0}$, contradicting $\|\mathbf{S}\mathbf{x}\| \approx \alpha \cdot \|\mathbf{x}\| > 0$. $\square$

So far, we have shown how to use lattices and sublattices to define finite Abelian groups. It is also possible to use finite Abelian groups to define lattices.

PROPOSITION 2.11. *Let $G$ be a finite Abelian group, and let $g_1, \ldots, g_n$ be a sequence of elements of $G$. Then, the set*

$$\Lambda(g_1, \ldots, g_n) = \left\{ \mathbf{x} \in \mathbb{Z}^n \colon \sum_{i=1}^{n} x_i g_i = 0 \right\}$$

*is a lattice, and its determinant satisfies $\det(\Lambda(g_1, \ldots, g_n)) \le \#G$, with equality if and only if $g_1, \ldots, g_n$ generate the entire group $G$.*

*Proof.* $\Lambda(g_1, \ldots, g_n)$ is a lattice because it is an additive subgroup of $\mathbb{Z}^n$. Let $G'$ be the subgroup generated by $g_1, \ldots, g_n$. Notice that the quotient $\mathbb{Z}^n / \Lambda(g_1, \ldots, g_n)$ is isomorphic to $G'$, with isomorphism given by $\phi([\mathbf{x}]) = \sum_i x_i g_i$. It follows that the size of $G'$ is $\det(\Lambda(g_1, \ldots, g_n))/\det(\mathbb{Z}^n) = \det(\Lambda(g_1, \ldots, g_n))$, and $\det(\Lambda(g_1, \ldots, g_n)) = \#G' \le \#G$. $\square$

**2.5. Statistical distance.** The statistical distance is a measure of how two probability distributions are far apart from each other, and it is a convenient tool in the analysis of randomized algorithms and reductions. In this subsection we define the statistical distance and state some simple facts that will be used in the analysis of the algorithms in this paper. All the properties of the statistical distance stated in this subsection are easily verified. For more details the reader is referred to [36, Chapter 8].

DEFINITION 2.12. *Let $X$ and $Y$ be two discrete random variables over a (countable) set $A$. The statistical distance between $X$ and $Y$ is the quantity*

$$\Delta(X,Y) = \frac{1}{2}\sum_{a\in A} |\Pr\{X=a\} - \Pr\{Y=a\}|.$$

We say that two random variables $X, Y$ are identically distributed (written $X \equiv Y$) if and only if $\Pr\{X=a\} = \Pr\{Y=a\}$ for every $a \in A$. The reader can easily check that the statistical distance satisfies the usual properties of distance functions, i.e., $\Delta(X,Y) \geq 0$ (with equality if and only if $X \equiv Y$), $\Delta(X,Y) = \Delta(Y,X)$, and $\Delta(X,Z) \leq \Delta(X,Y) + \Delta(Y,Z)$.

The following proposition shows that applying a (possibly randomized) function to two distributions does not increase the statistical distance.

PROPOSITION 2.13. *Let $X, Y$ be two random variables over a common set $A$. For any (possibly randomized) function $f$ with domain $A$, the statistical distance between $f(X)$ and $f(Y)$ is at most*

$$\Delta(f(X), f(Y)) \leq \Delta(X,Y). \tag{2.6}$$

Another useful property of the statistical distance is the following.

PROPOSITION 2.14. *Let $X_1, \ldots, X_k$ and $Y_1, \ldots, Y_k$ be two lists of totally independent random variables. Then*

$$\Delta((X_1,\ldots,X_k),(Y_1,\ldots,Y_k)) \leq \sum_{i=1}^{k} \Delta(X_i, Y_i). \tag{2.7}$$

The next proposition and corollary show how to use the statistical distance to estimate expectations and probabilities.

PROPOSITION 2.15. *If $X$ and $Y$ are random variables over set $A$ and $f\colon A \to [a,b]$ is a real valued function, then*

$$|\operatorname{Exp}[f(X)] - \operatorname{Exp}[f(Y)]| \leq |b-a| \cdot \Delta(X,Y). \tag{2.8}$$

As a corollary, we immediately obtain the following.

COROLLARY 2.16. *If $X$ and $Y$ are random variables over set $A$ and $p\colon A \to \{0,1\}$ is a predicate, then*

$$|\Pr[p(X)=1] - \Pr[p(Y)=1]| \leq \Delta(X,Y). \tag{2.9}$$

The following proposition gives a standard amplification technique that allows us to generate almost uniform samples from a group by adding a relatively small number of independent samples that are not too far from uniform.

PROPOSITION 2.17. *Let $(G,+)$ be a finite group, and let $A_1,\dots,A_k$ be $k$ independent (but possibly not identically distributed) random variables over $G$ such that $\Pr\{A_i = g\} \approx 1/\#G$ for all $i = 1,\dots,k$ and any $g \in G$. Then, for any $g \in G$,*

$$\Pr\left\{\sum_{i=1}^{k} A_i = g\right\} \in \frac{1}{\#G} \cdot \left[1 \pm \frac{1}{2^k}\right].$$

*In particular, the statistical distance between the sum $A = \sum_{i=1}^{k} A_i$ and the uniform distribution $U$ over $G$ is at most*

$$\Delta\left(\sum_{i=1}^{k} A_i, U\right) \leq \frac{1}{2^{k+1}}.$$

The next proposition gives a way to estimate how much a given distribution is affected by conditioning.

PROPOSITION 2.18. *For any random variable $X$ over set $A$, and event $Y$, the statistical distance between distribution $X$ and the conditional distribution of $X$ given $Y$ is exactly half the expected relative additive error of $Y$ given $X$; i.e.,*

$$\Delta(X, (X|Y)) = \frac{1}{2} \operatorname*{Exp}_{X}\left[\left|\frac{\Pr\{Y \mid X\}}{\Pr\{Y\}} - 1\right|\right] \leq \frac{1}{2} \max_{a \in A}\left|\frac{\Pr\{Y|X = a\}}{\Pr\{Y\}} - 1\right|.$$

**2.6. Iterative algorithms and reductions.** Many lattice algorithms work by first computing a relatively poor solution to the problem in question, and then iteratively improving it until a solution meeting some desired condition is found. Examples of such iterative algorithms are the LLL basis reduction algorithm [27] and the Ajtai worst- to average-case reduction and variants [2, 11, 35, 34]. Many other fundamental algorithms can also be described as an iterative process; e.g., Euclid's algorithm starts from two input numbers $a, b$ and iteratively computes smaller and smaller numbers until the greatest common divisor of $a$ and $b$ is found.

The high level structure of many iterative algorithms is the same, and it can be formulated abstractly without any reference to the specific problem in question. Although the method is now standard and has been repeatedly used to solve many lattice problems, it has never been explicitly formulated. In order to avoid unnecessary repetitions and highlight both the similarities and differences among all these algorithms, below we give an abstract formulation of this iterative method and present a general proof that the method implies standard polynomial time algorithms and reductions.

Computational problems can be abstractly defined by giving a binary relation consisting of all problem-solution pairs.

DEFINITION 2.19. *The language associated to a binary relation $R$ is the set $L_R$ of all strings $x$ such that $(x,w) \in R$ for some $w$. The problem defined by relation $R$ is as follows: given a string $x \in L_R$, find a $w$ such that $(x,w) \in R$. For any $(x,w) \in R$, we say that $w$ is a solution to problem $x$.*

Since no polynomial time algorithm (or reduction) can possibly output a solution $w$ of size superpolynomial in $|x|$, it is usually assumed that there exists a polynomial $p$ such that $|w| \leq p(|x|)$ for all $(x,w) \in R$. This is the case for all problems considered in this paper. However, we remark that in general relation $R$ is not required to be polynomial time computable. For example, in the $\gamma$ approximate SVP, membership in the associated relation $R = \{(\mathbf{B},\mathbf{v})\colon \mathbf{v} \in \mathcal{L}(\mathbf{B}) \setminus \{\mathbf{0}\} \wedge \|\mathbf{v}\| \leq \gamma \cdot \lambda_1(\mathcal{L}(\mathbf{B}))\}$ is probably not polynomial time computable for $\gamma < \sqrt{2}$ (unless NP = RP [33]). If membership in $R$ is polynomial time computable (and there exists a polynomial $p$ such that $|w| \leq p(|x|)$ for all $(x, w) \in R$), then $R$ is an NP-relation.

A deterministic algorithm $\mathcal{F}$ solves problem $R$ if for any $x \in L_R$ the output $w = \mathcal{F}(x)$ satisfies $(x, w) \in R$. If $\mathcal{F}$ is a randomized algorithm, then we say that $\mathcal{F}$ solves $R$ *in the worst case* with probability $p$ (possibly a function of the input length); if for any $x \in L_R$ the probability (over the internal randomness of $\mathcal{F}$ alone) that $(x, \mathcal{F}(x)) \in R$ is at least $p(|x|)$. Algorithm $\mathcal{F}$ solves $R$ *on the average* if $(x, \mathcal{F}(x)) \in R$ with probability at least $p(|x|)$, when the probability is computed over the internal randomness of $\mathcal{F}$ and the random selection of the input $x$ according to some specified distribution over all strings in $L_R$ of length $|x|$.

For all worst-case problems considered in this paper, given two tentative solutions $w_0, w_1$ for a problem $x$, it is possible to efficiently select the "best" of the two; i.e., there is a polynomial time algorithm that selects a $w_i$ such that if $(x, w_0) \in R$ or $(x, w_1) \in R$, then $(x, w_i) \in R$. It follows that any algorithm that solves the problem with nonnegligible probability $p(|x|) \geq |x|^{-O(1)}$ can be easily transformed into an algorithm that solves the problem with probability exponentially close to 1, by repeatedly executing the basic algorithm a polynomial (e.g., $|x| \ / \ p(|x|)$) number of times and selecting the best solution. So, we describe any such algorithm as solving the problem with *high probability*, without explicitly stating the exact value of the success probability. Notice, however, that this applies only to (randomized) algorithms that solve a problem in the *worst case*. The success probability amplification technique we just described may not work when applied to an algorithm that solves a problem with nonnegligible probability, but only *on the average*, when the input is chosen at random.

Before we give the general definition of iterative reduction (or algorithm), we illustrate it with a familiar example. Consider Euclid's algorithm. The input is a pair of numbers $x = (a, b)$, and we want to find the greatest common divisor $w = \gcd(a, b)$. The algorithm works iteratively, maintaining at each iteration a pair $s = (s_0, s_1)$ such that $s_0 \geq s_1$ and $\gcd(s_0, s_1) = w$. At every iteration, if $s_1 \neq 0$, the state is updated from $(s_0, s_1)$ to $(s_1, s_0 \bmod s_1)$. Polynomial time termination is guaranteed because at every iteration the function $f(s_0, s_1) = s_0 \cdot s_1 + 1$ decreases at least by a factor 2. When $s_1 = 0$ no more progress is possible, and the algorithm terminates with output $s_0$. In the case of lattice problems, the input is usually a lattice basis $x = \mathbf{B}$, and the algorithm maintains a set of linearly independent vectors $s = \mathbf{S} \subset \mathcal{L}(\mathbf{B})$, or a basis for the input lattice $\mathcal{L}(\mathbf{S}) = \mathcal{L}(\mathbf{B})$. Typically, the goal is to find a set of short vectors $\mathbf{S}$. This set is found by initially setting $\mathbf{S}$ to the input basis and then iteratively applying an algorithm that on input $\mathbf{B}$ and $\mathbf{S}$ outputs a better set $\mathbf{S}'$. This general idea is formalized in the following definition. (See Figure 1 for a pictorial representation of the intended use of iterative reductions and the proof of Theorem 2.21 below for further explanations.)

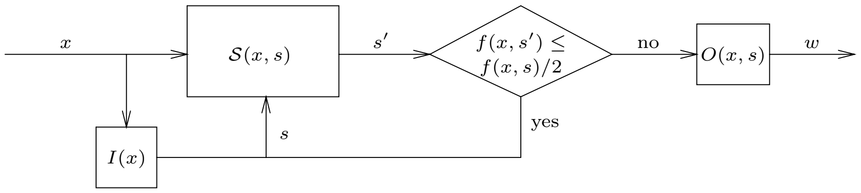


FIG. 1. *Iterative reduction.*

DEFINITION 2.20. *An* iterative reduction *from problem $P$ to a target problem $P'$ is a tuple $(R, f, I, O, \mathcal{S})$, where the following hold:*

(i) *$R$ is a relation (defining the set $R_x = \{s\colon (x, s) \in R\}$ of valid internal states) such that $|s| \le p(|x|)$ for some polynomial $p$ and all $(x, s) \in R$.*

(ii) *$f : R \to \mathbb{Q}^+$ (the* progress function*) is a polynomial time computable function mapping pairs $(x, s) \in R$ to positive rational numbers $f(x, s) \in \mathbb{Q}^+$.*

(iii) *$I$ (the* initialization function*) is a polynomial time computable function mapping each problem instance $x \in L_R$ to an initial state $I(x)$ satisfying $(x, I(x)) \in R$.*

(iv) *$O$ (the* output function*) is a polynomial time computable function mapping valid pairs $(x, s) \in R$ to tentative solutions $O(x, s)$, possibly satisfying $(x, O(x, s)) \in P$.*

(v) *$\mathcal{S}^{(\cdot)}$ (the* iterative step*) is a polynomial time oracle algorithm such that for any oracle $\mathcal{F}$ solving problem $P'$, and for all $(x, s) \in R$, the output $s' = \mathcal{S}^{\mathcal{F}}(x, s)$ satisfies*

(a) *$(x, s') \in R$; and*

(b) *if $P(x, O(x, s))$ is false, then $f(x, s') \le f(x, s)/2$.*

The following theorem shows that iterative reductions easily imply the existence of standard Cook reductions. We remark that the notion of iterative reduction formulated in Definition 2.20 and used in Theorem 2.21 below considers both $P$ and $P'$ as *worst-case* problems. The definition can be easily extended to the case where $P'$ is an average-case problem; however, this is not needed in this paper. In section 6 we will show that the iterative reduction implicit in Ajtai's worst-case to average-case connection and variants can be easily factored into an iterative reduction between two worst-case problems and a worst- to average-case reduction that involves no iteration.

THEOREM 2.21. *If there is an iterative reduction from problem $P$ to problem $P'$, then there is a polynomial time (Cook) reduction from $P$ to $P'$.*

*Proof.* Let $(R, f, I, O, \mathcal{S})$ be an iterative reduction from $P$ to $P'$. We define a standard polynomial time reduction between the two problems. The reduction works as follows (see Figure 1):

1. On input $x$, compute $s = I(x)$.
2. Compute $\mathcal{S}^{\mathcal{F}}(x, s) = s'$ and check if $f(x, s') \le f(x, s)/2$.
3. If the check succeeds, replace $s$ with $s'$ and repeat the previous step.
4. If the check fails, then terminate and output $w = O(x, s)$.

It is immediate to verify that the algorithm satisfies the invariant $(x, s) \in R$ at all iterations. Moreover, the termination condition $f(x, s') > f(x, s)/2$ implies $P(x, O(x, s))$, and therefore the final output is a correct solution. We need to prove that the running time is polynomial in $|x|$. Notice that since $(x, s) \in R$, $|s|$ is bounded by a fixed polynomial in $|x|$ in all iterations, and all steps can be performed in polynomial time. We need to bound the number of iterations. Let $p$ be a polynomial such that $|s| \le p(|x|)$ for all $(x, s) \in R$, and let $q$ be a polynomial bounding the running time of $f$. It follows that the initial value of $f(x, s)$ is at most $2^{q(|x|+p(|x|))}$ and that $f(x, s)$ is always at least $2^{-q(|x|+p(|x|))}$. Since at every iteration $f(x, s)$ decreases by a factor 2, the maximum number of iterations is at most $2q(|x| + p(|x|))$, which is polynomial in $|x|$. $\square$

**3. Covering radius and uniform radius.** Let $\mathcal{L}(\mathbf{B})$ be an $n$-dimensional lattice, and let $\mathcal{Q}$ be a convex body in $\mathbb{R}^n$. It can be shown that if we consider a randomly shifted copy of the body $\mathcal{Q} + \mathbf{x}$ (where $\mathbf{x}$ is chosen uniformly at random[14] ), then the expected number of lattice points in $\mathcal{Q} + \mathbf{x}$ equals exactly

$$\operatorname*{Exp}_{\mathbf{x}}\left[\#(\mathcal{L}(\mathbf{B}) \cap (\mathcal{Q} + \mathbf{x}))\right] = \frac{\operatorname{vol}(\mathcal{Q})}{\det(\mathcal{L}(\mathbf{B}))}.$$

In particular, if $\mathcal{Q}$ is a sphere of radius $r$, then

$$\operatorname*{Exp}_{\mathbf{x}}\left[\#(\mathcal{L}(\mathbf{B}) \cap \mathcal{B}(\mathbf{x}, r))\right] = \frac{\operatorname{vol}(\mathcal{B}(r))}{\det(\mathcal{L}(\mathbf{B}))}.$$

This corresponds to the intuition that the determinant $\det(\mathcal{L}(\mathbf{B}))$ is the inverse of the density of lattice points in space. Notice that the actual number of lattice points in a specific $\mathcal{Q}$ may deviate arbitrarily from the expectation, even for the special case of spherical $\mathcal{Q}$. Consider, for example, a lattice generated by two orthogonal vectors $\mathbf{e}_1$ and $D\mathbf{e}_2$, where $D$ is a large constant. Notice that the determinant of the lattice is $D$, so on the average we would expect to find $\operatorname{vol}(\mathcal{Q})/D$ lattice points inside $\mathcal{Q}$. Now, let $\mathcal{Q} = \mathcal{B}(\mathbf{x}, \sqrt{D})$ be the open disc of radius $\sqrt{D}$. The area of $\mathcal{Q}$ is $\operatorname{vol}(\mathcal{Q}) = \pi D$, so on the average we would expect to find $\pi$ lattice points in $\mathcal{Q}$. However, if $\mathbf{x} = 0$, the number of lattice points in $\mathcal{Q}$ is $2\lceil\sqrt{D}\rceil - 1$. Even worse, if $\mathbf{x} = (D/2)\mathbf{e}_2$, then $\mathcal{Q}$ does not contain any lattice point at all.

We define the *uniform radius* of a lattice as the smallest value $r = \zeta(\mathcal{L}(\mathbf{B}))$ such that any sphere $\mathcal{B}(\mathbf{x}, r)$ contains a number of lattice points close to the expected value.

DEFINITION 3.1. *For any $n$-dimensional lattice $\mathcal{L}(\mathbf{B})$, the* uniform radius $\zeta(\mathcal{L}(\mathbf{B}))$ *is the smallest positive real $r$ such that*

$$\#(\mathcal{L}(\mathbf{B}) \cap \mathcal{B}(\mathbf{x}, r)) \approx \frac{\operatorname{vol}(\mathcal{B}(r))}{\det(\mathcal{L}(\mathbf{B}))}$$

*for any* $\mathbf{x} \in \operatorname{span}(\mathcal{L}(\mathbf{B}))$.

The following proposition shows that the uniform radius $\zeta(\mathcal{L}(\mathbf{B}))$ is at least as large as the covering radius $\rho(\mathcal{L}(\mathbf{B}))$. Later we will also show that the uniform radius is never much bigger than that.

PROPOSITION 3.2. *For any lattice $\mathcal{L}(\mathbf{B})$, $\rho(\mathcal{L}(\mathbf{B})) < \zeta(\mathcal{L}(\mathbf{B}))$.*

*Proof.* The proof is immediate because for $r \le \rho(\mathcal{L}(\mathbf{B}))$ any sphere of radius $r$ centered in a deep hole (i.e., a point in space at distance $\rho(\mathcal{L}(\mathbf{B}))$ from the lattice) does not contain any lattice point. ☐

The uniform radius can be used to estimate the number of lattice points contained in a sphere. Later in this paper, we need to estimate the number of lattice points inside arbitrary convex bodies. So, we generalize the definition of the uniform radius to arbitrary convex bodies.

DEFINITION 3.3. *For any $n$-dimensional lattice $\mathcal{L}(\mathbf{B})$, the* generalized uniform radius $\hat{\zeta}(\mathcal{L}(\mathbf{B}))$ *is the smallest positive real $r$ such that for any convex body $\mathcal{Q}$ containing a sphere $\mathcal{B}(\mathbf{x}, r) \subseteq \mathcal{Q}$ of radius $r$, the number of lattice points inside the body satisfies*

$$\#(\mathcal{L}(\mathbf{B}) \cap \mathcal{Q}) \approx \frac{\operatorname{vol}(\mathcal{Q})}{\det(\mathcal{L}(\mathbf{B}))}.$$

[14]Intuitively, we would like to choose $\mathbf{x}$ uniformly at random from $\mathbb{R}^n$, but this is not possible because $\mathbb{R}^n$ has infinite measure. This problem is easily solved observing that it is enough to choose $\mathbf{x}$ uniformly at random from the fundamental region $\mathcal{P}(\mathbf{B})$ of the lattice, because the lattice repeats identically when translated by $\mathbf{Bx}$ for $\mathbf{x} \in \mathbb{Z}^n$.

Clearly, the generalized uniform radius is at least as large as the uniform radius: for any lattice $\mathcal{L}(\mathbf{B})$, $\zeta(\mathcal{L}(\mathbf{B})) \leq \hat{\zeta}(\mathcal{L}(\mathbf{B}))$. In particular, $\hat{\zeta}(\mathcal{L}(\mathbf{B}))$ is always larger than the covering radius $\rho(\mathcal{L}(\mathbf{B}))$. We bound the (generalized) uniform radius from above and show that for any lattice $\mathcal{L}(\mathbf{B})$, the (generalized) uniform radius is not much larger than the covering radius. Specifically, we show that $\hat{\zeta}(\mathcal{L}(\mathbf{B})) = O(n) \cdot \rho(\mathcal{L}(\mathbf{B}))$. A similar result was proved by Dyer, Frieze, and Kannan in [14] for the special case of $\mathcal{L}(\mathbf{B}) = \mathbb{Z}^n$. We observe that the proof of [14] is a general volume argument and does not use any special property of lattice $\mathbb{Z}^n$. So, it can be easily adapted to arbitrary lattices. Below we recall two simple geometric lemmas proved in [14] and then use them to prove the bound on $\hat{\zeta}(\mathcal{L}(\mathbf{B}))$.

Lemma 3.4 (see [14, Proposition 1]). *Suppose $\mathcal{Q}$ is a convex body in $\mathbb{R}^n$ containing the unit ball $\mathcal{B}(1)$, and let $\epsilon > 0$ be any positive real. Then all points within distance $\epsilon$ from $\mathcal{Q}$ belong to $(1+\epsilon)\mathcal{Q}$.*

Lemma 3.5 (see [14, Proposition 2]). *Suppose $\mathcal{Q}$ is a convex body in $\mathbb{R}^n$ containing the unit ball $\mathcal{B}(1)$, and let $0 < \epsilon \leq 1$. Then, all points within distance $\epsilon$ from $(1-\epsilon)\mathcal{Q}$ belong to $\mathcal{Q}$.*

We can now prove the bound on the uniform radius in terms of the covering radius.

Theorem 3.6. *For any $n$-dimensional lattice $\mathcal{L}(\mathbf{B})$,*

$$\hat{\zeta}(\mathcal{L}(\mathbf{B})) \leq 3n\rho(\mathcal{L}(\mathbf{B})).$$

*Proof.* Let $\mathcal{L}(\mathbf{B})$ be a full rank lattice in $\mathbb{R}^n$ with covering radius $\rho(\mathcal{L}(\mathbf{B}))$, and let $\mathcal{Q}$ be a convex body containing a sphere of radius $r = 3n\rho(\mathcal{L}(\mathbf{B}))$. We want to prove that $\#(\mathcal{L}(\mathbf{B}) \cap \mathcal{Q}) \approx \frac{\mathrm{vol}(\mathcal{Q})}{\det(\mathcal{L}(\mathbf{B}))}$. Let $\mathcal{L}(\mathbf{B}') = \mathcal{L}(\mathbf{B})/r$ and $\mathcal{Q}' = \mathcal{Q}/r$ be scaled versions of $\mathcal{L}(\mathbf{B})$ and $\mathcal{Q}$, and consider the set of points

$$S = \mathcal{L}(\mathbf{B}') \cap \mathcal{Q}' = \frac{\mathcal{L}(\mathbf{B}) \cap \mathcal{Q}}{r}.$$

Clearly, $\#S = \#(\mathcal{L}(\mathbf{B}') \cap \mathcal{Q}') = \#(\mathcal{L}(\mathbf{B}) \cap \mathcal{Q})$. We want to prove that

$$\#S \approx \frac{\mathrm{vol}(\mathcal{Q}')}{\det(\mathcal{L}(\mathbf{B}'))} = \frac{\mathrm{vol}(\mathcal{Q})}{\det(\mathcal{L}(\mathbf{B}))}.$$

Consider the union of all Voronoi cells $\mathcal{V}(\mathbf{x}, \mathcal{L}(\mathbf{B}'))$ with centers $\mathbf{x} \in S$. Notice that all points $\mathbf{y} \in \mathcal{V}(\mathbf{x}, \mathcal{L}(\mathbf{B}'))$ are within distance $\rho(\mathcal{L}(\mathbf{B}')) = \rho(\mathcal{L}(\mathbf{B}))/r$ from $\mathbf{x}$. Moreover, $\mathcal{Q}'$ contains a sphere of radius 1. Therefore, by Lemma 3.4, for all $\mathbf{x} \in S \subset \mathcal{Q}'$ and $\mathbf{y} \in \mathcal{V}(\mathbf{x}, \mathcal{L}(\mathbf{B}'))$, we have $\mathbf{y} \in \mathcal{Q}' \cdot (1+\rho(\mathcal{L}(\mathbf{B}))/r)$, i.e., $\mathcal{V}(\mathbf{x}, \mathcal{L}(\mathbf{B}')) \subseteq (1+\rho(\mathcal{L}(\mathbf{B}))/r) \cdot \mathcal{Q}'$. (Scaling, this time, was performed using as origin the center of the unit sphere contained in $\mathcal{Q}'$.) Since Voronoi cells are disjoint and have the same volume, we have

$$\begin{aligned}
\#S &= \frac{\sum_{\mathbf{x} \in S} \mathrm{vol}\left(\mathcal{V}(\mathbf{x}, \mathcal{L}(\mathbf{B}'))\right)}{\mathrm{vol}(\mathcal{V}(\mathcal{L}(\mathbf{B}')))} \\
&= \frac{\mathrm{vol}\left(\bigcup_{\mathbf{x} \in S} \mathcal{V}(\mathbf{x}, \mathcal{L}(\mathbf{B}'))\right)}{\mathrm{vol}(\mathcal{V}(\mathcal{L}(\mathbf{B}')))} \\
&\leq \frac{\mathrm{vol}(\mathcal{Q}' \cdot (1 + \rho(\mathcal{L}(\mathbf{B}))/r))}{\det(\mathcal{L}(\mathbf{B}'))} \\
&= \left(1 + \frac{\rho(\mathcal{L}(\mathbf{B}))}{r}\right)^n \frac{\mathrm{vol}(\mathcal{Q}')}{\det \mathcal{L}(\mathbf{B}')}.
\end{aligned}$$

Finally, using the assumption $r \geq 3n\rho(\mathcal{L}(\mathbf{B}))$, we get

$$\begin{aligned}\left(1+\frac{\rho(\mathcal{L}(\mathbf{B}))}{r}\right)^n &< \left(1+\frac{1}{3n-1}\right)^n \\ &= \frac{1}{\left(1-\frac{1}{3n}\right)^n} \\ &\leq \frac{1}{1-\frac{1}{3}} = \frac{3}{2}.\end{aligned}$$

This proves the upper bound $\#S \lesssim \mathrm{vol}(\mathcal{Q}')/\det\mathcal{L}(\mathbf{B}')$.

We now turn to the lower bound. Let $S'$ be the set of all lattice points $\mathbf{x} \in \mathcal{L}(\mathbf{B}')$ such that the Voronoi cell $\mathcal{V}(\mathbf{x}, \mathcal{L}(\mathbf{B}'))$ intersects $(1-\rho(\mathcal{L}(\mathbf{B}))/r)\mathcal{Q}'$. Notice that if $\mathcal{V}(\mathbf{x}, \mathcal{L}(\mathbf{B}'))$ intersects $(1-\rho(\mathcal{L}(\mathbf{B}))/r)\mathcal{Q}'$, then $\mathbf{x}$ must be within distance $\rho(\mathcal{L}(\mathbf{B}')) = \rho(\mathcal{L}(\mathbf{B}))/r$ from $(1-\rho(\mathcal{L}(\mathbf{B}))/r)\cdot\mathcal{Q}'$. So, by Lemma 3.5, $\mathbf{x} \in \mathcal{Q}'$. This proves that $S' \subseteq S$, and $\#S \geq \#S'$. Since Voronoi cells cover $\mathbb{R}^n$, $(1-\rho(\mathcal{L}(\mathbf{B}))/r)\mathcal{Q}'$ is fully contained in the union $\bigcup_{\mathbf{x}\in S'} \mathcal{V}(\mathbf{x}, \mathcal{L}(\mathbf{B}'))$, and

$$\begin{aligned}\#S' &= \frac{\sum_{\mathbf{x}\in S'} \mathrm{vol}(\mathcal{V}(\mathbf{x}, \mathcal{L}(\mathbf{B}')))}{\mathrm{vol}(\mathcal{V}(\mathcal{L}(\mathbf{B}')))} \\ &= \frac{\mathrm{vol}(\bigcup_{\mathbf{x}\in S'} \mathcal{V}(\mathbf{x}, \mathcal{L}(\mathbf{B}')))}{\mathrm{vol}(\mathcal{V}(\mathcal{L}(\mathbf{B}')))} \\ &\geq \frac{\mathrm{vol}((1-\rho(\mathcal{L}(\mathbf{B}))/r)\mathcal{Q}')}{\det(\mathcal{L}(\mathbf{B}'))} \\ &= \left(1-\frac{\rho(\mathcal{L}(\mathbf{B}))}{r}\right)^n \frac{\mathrm{vol}(\mathcal{Q}')}{\det(\mathcal{L}(\mathbf{B}'))}.\end{aligned}$$

Using the assumption $r \geq 3n\rho(\mathcal{L}(\mathbf{B}))$, we immediately get

$$\left(1-\frac{\rho(\mathcal{L}(\mathbf{B}))}{r}\right)^n > \left(1-\frac{1}{2n}\right)^n \geq 1-\frac{1}{2}.$$

This proves the lower bound $\#S \gtrsim \mathrm{vol}(\mathcal{Q}')/\det(\mathcal{L}(\mathbf{B}'))$ and completes the proof of the theorem. $\square$

Using inequality $\rho(\mathcal{L}(\mathbf{B})) \leq \sqrt{n}\cdot\lambda_n(\mathcal{L}(\mathbf{B}))/2$ from (2.3), we can bound $\hat{\zeta}(\mathcal{L}(\mathbf{B}))$ in terms of $\lambda_n(\mathcal{L}(\mathbf{B}))$:

$$\hat{\zeta}(\mathcal{L}(\mathbf{B})) \leq \frac{3}{2}n^{1.5}\lambda_n(\mathcal{L}(\mathbf{B})). \tag{3.1}$$

Similarly, using the transference theorem (2.4), we can bound $\hat{\zeta}(\mathcal{L}(\mathbf{B}))$ in terms of the length of the shortest nonzero vector in the dual lattice:

$$\hat{\zeta}(\mathcal{L}(\mathbf{B})) \leq \frac{3}{2}n^2\frac{1}{\lambda_1(\mathcal{L}(\mathbf{B})^*)}. \tag{3.2}$$

These bounds can be used to relate the average-case complexity of finding small solutions to random equations to the worst-case complexity of approximating SIVP or GapSVP.

It would be interesting to improve bounds (3.1) and (3.2). In particular, is it true that $\hat{\zeta}(\mathcal{L}(\mathbf{B})) = O(n)\cdot\lambda_n(\mathcal{L}(\mathbf{B}))$ for any $n$-dimensional lattice $\mathcal{L}(\mathbf{B})$? Is it true that $\hat{\zeta}(\mathcal{L}(\mathbf{B})) = O(n)/\lambda_1(\mathcal{L}(\mathbf{B})^*)$? Proving these improved bounds would immediately result in a reduction of the connection factor for SIVP by a factor $O(\sqrt{n})$ and for GapSVP by a factor $O(n)$.

**4. Easily decodable almost perfect lattices.** We are interested in lattices that have both good algorithmic and geometric properties. Algorithmically, we would like lattices where the CVP can be efficiently solved. Notice that, despite the NP-hardness of CVP, the CVP may be efficiently solvable for *specific* lattices. For example, in the integer lattice $\mathbb{Z}^n$, a lattice vector $\mathbf{x} \in \mathbb{Z}^n$ closest to a given target $\mathbf{t} \in \mathbb{Q}^n$ can be easily found rounding each coordinate of $\mathbf{t}$ to the closest integer $x_i = \lfloor t_i \rceil$. Since for any fixed dimension the CVP can be solved in polynomial time, in order to properly formulate this problem one needs to consider not a single lattice but an infinite sequence of lattices in increasing dimension. For simplicity, in the definition below we focus on full rank lattices, although this restriction is not necessary.

DEFINITION 4.1. *Let $\{\mathbf{L}_n\}_{n\geq 1}$ be a sequence of full rank lattices $\mathcal{L}(\mathbf{L}_n) \subseteq \mathbb{R}^n$. We say that the sequence $\{\mathbf{L}_n\}_{n\geq 1}$ is* easily decodable *if there exists a polynomial time algorithm* $\mathrm{CVP}_{\mathbf{L}}$ *such that for any $n \geq 1$ and $\mathbf{t} \in \mathbb{Q}^n$,* $\mathrm{CVP}_{\mathbf{L}}(\mathbf{t})$ *outputs a lattice vector in $\mathcal{L}(\mathbf{L}_n)$ closest to $\mathbf{t}$.*

The simplest example of an easily decodable sequence of lattices is given by the integer lattices $\mathbb{Z}^n$ defined by matrices $\mathbf{L}_n = \mathbf{I}_n$. Other easily decodable lattices considered in [12] are the *root lattices* $A_n$ and $D_n$ and their duals $D_n^*$ and $A_n^*$.[15]

From a geometric point of view, we would like the Voronoi cells of the lattice to be as spherical as possible. Remember that the Voronoi cell $\mathcal{V}(\mathcal{L}(\mathbf{L}))$ contains a sphere $\mathcal{B}(\lambda_1(\mathcal{L}(\mathbf{L}))/2)$ with radius equal to the packing radius and is completely contained in a sphere $\bar{\mathcal{B}}(\rho(\mathcal{L}(\mathbf{L})))$ with radius equal to the covering radius. So, the closer the covering radius is to the packing radius, the better the Voronoi cells are approximated by spheres. This motivates the following definition.

DEFINITION 4.2. *For any $\tau \geq 1$, a lattice $\mathcal{L}(\mathbf{L})$ is $\tau$-perfect if*

$$\rho(\mathcal{L}(\mathbf{L})) \leq \tau \cdot \left(\frac{\lambda_1(\mathcal{L}(\mathbf{L}))}{2}\right).$$

*For any function $\tau(n)$, a sequence of (full rank) lattices $\{\mathbf{L}_n\}_{n\geq 1}$ (where $n$ is the dimension of $\mathcal{L}(\mathbf{L}_n)$) is $\tau(n)$-perfect if $\mathcal{L}(\mathbf{L}_n)$ is $\tau(n)$-perfect for any $n \geq 1$. A sequence of (full rank) lattices $\{\mathbf{L}_n\}_{n\geq 1}$ is almost perfect if it is $\tau(n)$-perfect for some $\tau(n) = o(\sqrt{n})$.*

We are interested in sequences of lattices such that $\tau(n)$ is as small as possible. Moreover, we would like the lattices to be easily decodable. The integer lattice $\mathbb{Z}^n$, as well as all other sequences $A_n, A_n^*, D_n, D_n^*$ of easily decodable lattices considered in [12], are $\tau(n)$-perfect for $\tau(n) = \Theta(\sqrt{n})$, so they are not almost perfect. It is natural to ask if nontrivial easily decodable almost perfect lattices (i.e., $\tau(n)$-perfect lattices with $\tau(n) = o(\sqrt{n})$) exist, or if the almost perfectness and easy decodability requirements are incompatible.

In this section we start the algorithmic study of almost perfect lattices and give the first efficient construction of nontrivial easily decodable almost perfect lattices. Our lattices are $\tau(n)$-perfect for $\tau(n) = \sqrt{n \log\log n/\log n} = o(\sqrt{n})$. Although this is not a substantial improvement over $\tau(n) = \Theta(\sqrt{n})$ from a quantitative point of view, it is qualitatively interesting because it shows that nontrivial easily decodable almost perfect lattices exist.

We first present a construction of 3-perfect lattices such that the construction and the decoding algorithm run in exponential time $n^{O(n)}$. Then we show how to

[15]Conway and Sloane [12] also describe other efficient decoding algorithms for specific lattices, but $\mathbb{Z}^n, A_n, A_n^*, D_n, D_n^*$ are the only infinite sequences of lattices considered for which the problem of efficient decoding admits an interesting asymptotic formulation.

use small-dimensional lattices obtained using this construction to efficiently construct $O(\sqrt{n \log\log n / \log n})$-perfect lattices such that the CVP can be solved in polynomial time. The construction is based on the following simple lemma.

LEMMA 4.3. *For any lattice $\mathcal{L}(\mathbf{B})$, there exists a lattice vector $\mathbf{v} \in \mathcal{L}(\mathbf{B})$ such that* $\operatorname{dist}(\mathbf{v}/3, \mathcal{L}(\mathbf{B})) \geq (2/3)\rho(\mathcal{L}(\mathbf{B}))$. *In particular, if $\rho(\mathcal{L}(\mathbf{B})) \geq 3 \cdot \lambda_1(\mathcal{L}(\mathbf{B}))/2$, then* $\operatorname{dist}(\mathbf{v}/3, \mathcal{L}(\mathbf{B})) \geq \lambda_1(\mathcal{L}(\mathbf{B}))$.

*Proof.* Let $\mathbf{h}$ be a deep hole, i.e., a point in $\operatorname{span}(\mathcal{L}(\mathbf{B}))$ at distance $\rho(\mathcal{L}(\mathbf{B}))$ from $\mathcal{L}(\mathbf{B})$. Consider the point $3\mathbf{h}$, and let $\mathbf{v} \in \mathcal{L}(\mathbf{B})$ be a lattice point closest to $3\mathbf{h}$. By definition of the covering radius, it must be $\|\mathbf{v} - 3\mathbf{h}\| \leq \rho(\mathcal{L}(\mathbf{B}))$. Therefore, dividing by 3, we get $\|\mathbf{v}/3 - \mathbf{h}\| \leq \rho(\mathcal{L}(\mathbf{B}))/3$, and by the triangle inequality

$$\begin{aligned}\operatorname{dist}(\mathbf{v}/3, \mathcal{L}(\mathbf{B})) &\geq \operatorname{dist}(\mathbf{h}, \mathcal{L}(\mathbf{B})) - \operatorname{dist}(\mathbf{v}/3, \mathbf{h}) \\ &\geq \rho(\mathcal{L}(\mathbf{B})) - \frac{1}{3}\rho(\mathcal{L}(\mathbf{B})) \\ &= \frac{2}{3} \cdot \rho(\mathcal{L}(\mathbf{B})). \qquad \square\end{aligned}$$

We use the lemma to give an algorithmic construction of $\tau$-perfect lattices with $\tau < 3$. The following theorem is essentially an algorithmic variant of the proof of existence given in [40]. Both the procedure to build the lattice and the one to decode it run in time $n^{O(n)}$. It should be noted that for any $n$-dimensional lattice, in principle the CVP can always be solved in time $n^{O(n)}$ [23]. However, the algorithm of [23] for general lattices is rather complex. In the theorem below we show how to build a lattice $\mathcal{L}(\mathbf{B})$ together with some (polynomial size) auxiliary information $\mathbf{V}$ that allows us to solve the CVP in lattice $\mathcal{L}(\mathbf{B})$, still in time $n^{O(n)}$ as in [23], but with a much simpler algorithm.

THEOREM 4.4. *There is an algorithm running in time $n^{O(n)}$ that on input $n$ outputs an $n$-dimensional* 3*-perfect lattice $\mathbf{L}_n$. Moreover, the sequence of lattices $\{\mathbf{L}_n\}_{n\geq 1}$ is decodable in time $n^{O(n)}$; i.e., there is an algorithm* $\mathrm{CVP}_{\mathbf{L}}$ *running in time $n^{O(n)}$ that on input a vector $\mathbf{t} \in \mathbb{Q}^n$ outputs a lattice vector* $\mathrm{CVP}_{\mathbf{L}}(\mathbf{t}) \in \mathcal{L}(\mathbf{L}_n)$ *closest to* $\mathbf{t}$.

*Proof.* The algorithm starts from an arbitrary $n$-dimensional easily decodable lattice $\mathcal{L}(\mathbf{B}_0)$, e.g., the integer lattice $\mathcal{L}(\mathbf{B}_0) = \mathbb{Z}^n$ generated by the identity matrix $\mathbf{B}_0 = \mathbf{I}$. Notice that the closest vector in $\mathbb{Z}^n$ to a target $\mathbf{t}$ can be easily found by rounding each coordinate of $\mathbf{t}$ to the closest integer. Below we assume that $\mathbf{B}_0 = \mathbf{I}$ and, in particular, $\det(\mathcal{L}(\mathbf{B}_0)) = 1$ and $\lambda_1(\mathcal{L}(\mathbf{B}_0)) = 1$, but the construction works for any easily decodable lattice.

Starting from $\mathbf{B}_0$, we iteratively build a sequence of lattice bases $\mathbf{B}_i$ and auxiliary vectors $\mathbf{v}_i$ for $i = 1, \ldots, m$ for some $m = O(n \log n)$ to be determined. The final output are basis $\mathbf{B} = \mathbf{B}_m$ and set of vectors $\mathbf{V} = [\mathbf{v}_1, \ldots, \mathbf{v}_m]$. For each $k = 1, \ldots, m$, vector $\mathbf{v}_k$ and basis $\mathbf{B}_k$ are computed as follows:

1. For any vector $\mathbf{s} \in \{-1, 0, +1\}^n$, let $\mathbf{t} = (1/3)\mathbf{B}_{k-1}\mathbf{s}$ and compute the distance of $\mathbf{t}$ from the lattice $\mathcal{L}(\mathbf{B}_{k-1})$. (We will show below how this can be done in time $n^{O(1)} \cdot 3^k$.)
2. If for all $\mathbf{s} \in \{-1, 0, +1\}^n$, $\operatorname{dist}(\mathbf{t}, \mathcal{L}(\mathbf{B}_{k-1})) < 1$, then set $m = k - 1$, and terminate with output $\mathbf{B} = \mathbf{B}_{k-1}$ and $\mathbf{V} = [\mathbf{v}_1, \ldots, \mathbf{v}_{k-1}]$.
3. Otherwise (if $\operatorname{dist}(\mathbf{t}, \mathcal{L}(\mathbf{B}_{k-1})) \geq 1$ for some $\mathbf{s}$), proceed as follows. Notice that since $\operatorname{dist}(\mathbf{t}, \mathcal{L}(\mathbf{B}_{k-1})) > 0$, it must be $\mathbf{s} \neq \mathbf{0}$.
4. Let $i \in \{1, \ldots, n\}$ such that $s_i \neq 0$.
5. Set $\mathbf{v}_k = \mathbf{t}$.
6. Set $\mathbf{B}_k$ to the matrix obtained by replacing the $i$th vector in $\mathbf{B}_{k-1}$ with $\mathbf{v}_k$.

The algorithm uses a procedure to find closest vectors in lattice $\mathcal{L}(\mathbf{B}_k)$. We will show that the maximum number of iterations performed by the algorithm is $m \leq (n/2)\log_3 n = O(n \log n)$, and that for any $k$, the CVP in $\mathcal{L}(\mathbf{B}_k)$ can be solved in time $n^{O(1)} \cdot 3^k$. It follows that the total running time of the algorithm is

$$O(m \cdot n^{O(1)} \cdot 3^m) = n^{O(n)}$$

and that the CVP in $\mathcal{L}(\mathbf{B})$ can also be solved in time $n^{O(n)}$.

The correctness of the algorithm is based on the fact that for any $k$,

(i) vector $3\mathbf{v}_k$ belongs to the lattice $\mathcal{L}(\mathbf{B}_{k-1})$,
(ii) $\mathbf{B}_k$ is a basis for the lattice generated by $[\mathbf{B}_{k-1}|\mathbf{v}_k]$,
(iii) the shortest vector in $\mathcal{L}(\mathbf{B}_k)$ has length 1.

The first property immediately follows by construction. For the second property, it is clear that $\mathcal{L}(\mathbf{B}_k)$ is a subset of $\mathcal{L}([\mathbf{B}_{k-1}|\mathbf{v}_k])$. In order to prove $\mathcal{L}(\mathbf{B}_k) = \mathcal{L}([\mathbf{B}_{k-1}|\mathbf{v}_k])$ we need only to show that the $i$th vector of $\mathbf{B}_{k-1}$ (namely, $\mathbf{B}_{k-1}\mathbf{e}_i$) belongs to $\mathcal{L}(\mathbf{B}_k)$. Notice that $3\mathbf{v}_k = \mathbf{B}_{k-1}\mathbf{s} = \sum_j s_j \mathbf{B}_{k-1}\mathbf{e}_j$. So, $s_i \cdot \mathbf{B}_{k-1}\mathbf{e}_i = 3\mathbf{v}_k - \sum_{j\neq i} s_j \mathbf{B}_{k-1}\mathbf{e}_j = 3\mathbf{B}_k\mathbf{e}_i - \sum_{j\neq i} s_j \mathbf{B}_k \mathbf{e}_j$ belongs to $\mathcal{L}(\mathbf{B}_k)$. Since $s_i = \pm 1$, also $\mathbf{B}_{k-1}\mathbf{e}_i = \pm(s_i \cdot \mathbf{B}_{k-1}\mathbf{e}_i)$ belongs to $\mathcal{L}(\mathbf{B}_k)$. Now, let us get to the third property. Consider any nonzero vector in $\mathcal{L}(\mathbf{B}_k)$. Since $\mathcal{L}(\mathbf{B}_k) = \mathcal{L}([\mathbf{B}_{k-1}|\mathbf{v}_k])$, any such a vector can be written as $\mathbf{B}_{k-1}\mathbf{x} + \mathbf{v}_k \cdot y$. Moreover, since $3\mathbf{v}_k \in \mathcal{L}(\mathbf{B}_{k-1})$, we can assume without loss of generality that $y \in \{-1, 0, +1\}$. So, the length of $\mathbf{B}_{k-1}\mathbf{x} + \mathbf{v}_k \cdot y$ is at least the minimum of $\lambda_1(\mathcal{L}(\mathbf{B}_{k-1}))$ (if $y = 0$) or $\mathrm{dist}(\pm\mathbf{v}_k, \mathcal{L}(\mathbf{B}_{k-1}))$ (if $y = \pm 1$). But $\lambda_1(\mathcal{L}(\mathbf{B}_{k-1})) \geq 1$ by induction, and $\mathrm{dist}(\pm\mathbf{v}_k, \mathcal{L}(\mathbf{B}_{k-1})) = \mathrm{dist}(\mathbf{v}_k, \mathcal{L}(\mathbf{B}_{k-1})) \geq 1$ by construction. It follows that $\lambda_1(\mathcal{L}(\mathbf{B}_k)) \geq 1$.

It is also easy to see that for any $k$, the determinant of lattice $\mathcal{L}(\mathbf{B}_k)$ equals $\det(\mathcal{L}(\mathbf{B}_k)) = 3^{-k}\det(\mathcal{L}(\mathbf{B}_0)) = 3^{-k}$ because each $\mathbf{B}_k$ can be obtained from $\mathbf{B}_{k-1}$ by first performing some elementary integer column operations, and then dividing a column by 3. We can now prove that the algorithm performs at most $m = O(n \log n)$ iterations. Since $\lambda_1(\mathcal{L}(\mathbf{B}_k)) = 1$ and $\det(\mathcal{L}(\mathbf{B}_k)) = 3^{-k}$, by Minkowski's first theorem (2.2),

$$1 = \lambda_1(\mathcal{L}(\mathbf{B}_k)) \leq \sqrt{n}\det(\mathcal{L}(\mathbf{B}_k))^{1/n} = \sqrt{n}3^{-(k/n)}.$$

It follows that

$$k \leq (1/2)n\log_3 n = O(n \log n)$$

is an upper bound on the maximum number of iterations. (It can also be shown by a volume argument that $m = \Theta(n \log n)$ iterations are required in order to reach termination.)

Next we prove that upon termination $\rho(\mathcal{L}(\mathbf{L}_n)) < 3 \cdot \lambda_1(\mathcal{L}(\mathbf{L}_n))/2$. We show that if $\rho(\mathcal{L}(\mathbf{L}_n)) \geq (3/2)\cdot\lambda_1(\mathcal{L}(\mathbf{L}_n))$, then the algorithm certainly performs one more iteration. By Lemma 4.3, if $\rho(\mathcal{L}(\mathbf{L}_n)) \geq (3/2)\lambda_1(\mathcal{L}(\mathbf{L}_n))$, then there exists a vector $\mathbf{v} = \mathbf{B}_{k-1}\mathbf{x} \in \mathcal{L}(\mathbf{B}_{k-1})$ such that

$$\mathrm{dist}(\mathbf{v}/3, \mathcal{L}(\mathbf{B}_{k-1})) \geq \lambda_1(\mathcal{L}(\mathbf{B}_{k-1})) \geq 1.$$

Let $\mathbf{s} \in \{-1, 0, +1\}^n$ be such that $\mathbf{s} \equiv \mathbf{x} \pmod 3$, i.e., $(\mathbf{s} - \mathbf{x})/3 \in \mathbb{Z}^n$. We claim that the distance of $\mathbf{t} = (1/3)\mathbf{B}_{k-1}\mathbf{s}$ from the lattice $\mathcal{L}(\mathbf{B}_{k-1})$ is at least 1. Notice that

$$\mathbf{t} = (1/3)\mathbf{B}_{k-1}\mathbf{s} = \mathbf{B}_{k-1}\mathbf{x}/3 + \mathbf{B}_{k-1}(\mathbf{s} - \mathbf{x})/3 \in \mathbf{v}/3 + \mathcal{L}(\mathbf{B}_{k-1}).$$

It follows that $\operatorname{dist}(\mathbf{t}, \mathcal{L}(\mathbf{B}_{k-1})) = \operatorname{dist}(\mathbf{v}/3, \mathcal{L}(\mathbf{B}_{k-1})) \geq 1$, and therefore the algorithm does not terminate at iteration $k$.

We conclude the proof of the theorem by giving a simple algorithm to solve the CVP in $\mathcal{L}(\mathbf{B}_k)$ in time $n^{O(1)} \cdot 3^k \leq n^{O(n)}$. Notice that any lattice point in $\mathcal{L}(\mathbf{B}_k)$ can be written as $\mathbf{B}_0\mathbf{x} + [\mathbf{v}_1, \ldots, \mathbf{v}_k]\mathbf{y}$, where $\mathbf{x} \in \mathbb{Z}^n$ and $\mathbf{y} \in \{-1, 0, +1\}^k$. So, in order to find the lattice point closest to some target $\mathbf{t}$, we can consider all vectors of the form $\mathbf{t} - [\mathbf{v}_1, \ldots, \mathbf{v}_k]\mathbf{y}$ and compute their distance from $\mathcal{L}(\mathbf{B}_0)$. Let $\mathbf{y}$ be such that $\operatorname{dist}(\mathbf{t} - [\mathbf{v}_1, \ldots, \mathbf{v}_k]\mathbf{y}, \mathcal{L}(\mathbf{B}_k))$ is minimized, and let $\mathbf{B}_0\mathbf{x}$ be the lattice vector closest to $\mathbf{t} - [\mathbf{v}_1, \ldots, \mathbf{v}_k]\mathbf{y}$. The lattice vector in $\mathcal{L}(\mathbf{B}_k)$ closest to $\mathbf{t}$ is $\mathbf{B}_0\mathbf{x} + [\mathbf{v}_1, \ldots, \mathbf{v}_k]\mathbf{y}$. $\square$

The theorem gives an algorithmic construction of almost perfect lattices and an algorithm to solve the CVP; however, the running time is huge. The next theorem shows how to use these lattices for small values of $n$ to get a construction that runs in polynomial time.

THEOREM 4.5. *There exists a family of $\tau(n)$-perfect easily decodable lattices with $\tau(n) = O(\sqrt{n \log\log n / \log n})$.*

*Proof.* In order to keep the construction polynomial in $n$, we use Theorem 4.4 to build a 3-perfect lattice $\mathbf{M}$ in dimension $m = \log n / \log\log n$. Notice that such a lattice can be constructed in time

$$2^{O(m \log m)} = 2^{O(\log n \log(\log n / \log\log n) / \log\log n)} = n^{O(1)}$$

polynomial in $n$. Moreover, the CVP in this lattice can also be solved in polynomial time $2^{O(m \log m)} = n^{O(1)}$.

Now set $\mathcal{L}(\mathbf{L}_n)$ to the direct sum of $(n/m)$ copies of $\mathcal{L}(\mathbf{M})$, i.e., the lattice generated by the block diagonal matrix with $n/m$ blocks, all equal to $\mathbf{M}$. The lattice vector in $\mathcal{L}(\mathbf{L}_n)$ closest to a target $\mathbf{t}$ is easily found by breaking $\mathbf{t}$ into $n/m$ blocks, each with $m$ coordinates in it, and finding the $\mathcal{L}(\mathbf{M})$ vector closest to each block. Moreover, the length of the shortest nonzero vector in $\mathcal{L}(\mathbf{L}_n)$ is $\lambda_1(\mathcal{L}(\mathbf{M}))$ because vectors from different copies of $\mathbf{M}$ are orthogonal. Finally, the covering radius of $\mathcal{L}(\mathbf{L}_n)$ is $\sqrt{n/m}$ times $\rho(\mathcal{L}(\mathbf{M}))$. So, $\mathcal{L}(\mathbf{L}_n)$ is $\tau(n)$-perfect for

$$\tau(n) = \frac{\sqrt{n/m}\rho(\mathcal{L}(\mathbf{M}))}{\lambda_1(\mathcal{L}(\mathbf{M}))/2} \leq 3\sqrt{n/m} = O(\sqrt{n \log\log n / \log n}). \qquad \square$$

**5. A generalized class of random equations.** In this section we define a class of random equations that generalizes Ajtai's one. Ajtai's problem can be described as finding a small (e.g., with respect to the bound proved in Theorem 5.5 below) nonzero integer solution to a homogeneous linear equation in $m(n)$ variables with coefficients chosen uniformly at random from the group $G_n = \mathbb{Z}^n_{q(n)}$ of $n$-dimensional vectors modulo $q(n)$, for appropriately chosen functions $q(n)$ and $m(n)$. Here we consider equations with coefficients in a group $G_n$ possibly different from $\mathbb{Z}^n_{q(n)}$. In general, we define the following problem.

DEFINITION 5.1. *Let $\{G_n\}$ be a sequence of finite Abelian groups, and let $m(n)$ and $\beta(n)$ be two arbitrary (polynomial time computable) functions. For any $m(n)$-dimensional vector $\mathbf{g} = [g_1, \ldots, g_{m(n)}]^T \in G_n^{m(n)}$, define the set of solutions to the associated homogeneous linear equation*

$$\Lambda(\mathbf{g}) = \left\{ \mathbf{z} \in \mathbb{Z}^{m(n)} \colon \sum_{i=1}^{m(n)} z_i \cdot g_i = 0 \right\}. \tag{5.1}$$

*The* homogeneous small integer solution *problem* $\mathrm{HSIS}_{G,m,\beta}$ *(in the $\ell_2$ norm) is as follows: given a (random) homogeneous linear equation $\mathbf{g} \in G_n^{m(n)}$, find a nonzero solution $\mathbf{z} \in \Lambda(\mathbf{g}) \setminus \{\mathbf{0}\}$ of $\ell_2$ norm at most $\|\mathbf{z}\| \leq \beta(n)$.*

Of course, the problem is interesting only when a solution of length at most $\beta(n)$ exists. Below we define a family of groups (that includes Ajtai's groups as a special case) and give conditions under which a solution of length at most $\beta(n)$ is guaranteed to exist.

Our groups are parametrized by an easily decodable family of lattices $\{\mathcal{L}(\mathbf{L}_n)\}_{n>0}$ and a function $\alpha(n)$, and each group $G_n = \mathcal{G}(\mathcal{L}(\mathbf{L}_n), \alpha(n))$ is defined as the quotient of $\mathcal{L}(\mathbf{L}_n)$ modulo an appropriately chosen full dimensional sublattice $\mathcal{L}(\mathbf{M}_n) \subseteq \mathcal{L}(\mathbf{L}_n)$.

Definition 5.2. *For any easily decodable family of lattices $\{\mathcal{L}(\mathbf{L}_n)\}_{n>0}$ (with decoding algorithm $\mathrm{CVP}_{\mathbf{L}}$) and function $\alpha(n)$ satisfying $\alpha(n) \geq 2\sqrt{n}\rho(\mathcal{L}(\mathbf{L}_n))$, define the sequence of quotient groups*

$$G_n = \mathcal{G}(\mathcal{L}(\mathbf{L}_n), \alpha(n)) = \mathcal{L}(\mathbf{L}_n)/\mathcal{L}(\mathbf{M}_n), \tag{5.2}$$

*where for any $n > 0$, $\mathcal{L}(\mathbf{M}_n)$ is the full rank sublattice of $\mathcal{L}(\mathbf{L}_n)$ obtained by applying Lemma* 2.10 *with value $\alpha(n)$ and decoding procedure* $\mathrm{CVP}_{\mathbf{L}}$.

From Lemma 2.10 we immediately obtain the following corollary.

Corollary 5.3. *For any family of lattices $\{\mathcal{L}(\mathbf{L}_n)\}_{n>0}$ and function $\alpha(n)$ satisfying the conditions in Definition* 5.2, *the groups $\mathcal{G}(\mathcal{L}(\mathbf{L}_n), \alpha(n)) = \mathcal{L}(\mathbf{L}_n)/\mathcal{L}(\mathbf{M}_n)$ can be computed in time polynomial in $n$ and the matrix $\mathbf{M}_n$ satisfies*

$$\forall \mathbf{x} \in \mathbb{R}^n. \|\mathbf{M}_n\mathbf{x}\| \approx \alpha(n) \cdot \|\mathbf{x}\|.$$

*Proof.* Matrix $\mathbf{M}_n$ is polynomial time computable because lattice $\mathcal{L}(\mathbf{L}_n)$ is easily decodable, so the decoding procedure $\mathrm{CVP}_{\mathbf{L}}$ runs in polynomial time. In order to bound $\|\mathbf{M}_n\mathbf{x}\|$, simply observe that $\mathcal{L}(\mathbf{L}_n)$, $\alpha(n)$ and $\mathrm{CVP}_{\mathbf{L}}$ satisfy the conditions of Lemma 2.10 because $\alpha(n) \geq 2\sqrt{n}\rho(\mathcal{L}(\mathbf{L}_n)) \geq 2\sqrt{n}\,\mathrm{dist}(\mathbf{x}, \mathrm{CVP}_{\mathbf{L}}(\mathbf{x}))$ for all $\mathbf{x} \in \mathbb{R}^n$. $\square$

Notice that if $\mathcal{L}(\mathbf{L}_n) = \mathbb{Z}^n$ is the integer lattice and $\alpha(n) = q(n)$, then Definition 5.2 gives matrix $\mathbf{M}_n = q(n) \cdot \mathbf{I}$ and Ajtai's group $\mathcal{G}(\mathcal{L}(\mathbf{L}_n), \alpha(n)) = \mathbb{Z}^n_{q(n)}$ as a special case. We will see that this choice of group $G_n$ is not the best possible for our analysis, and setting $\mathcal{L}(\mathbf{L}_n)$ to an almost perfect lattice leads to better results. In the rest of this section, we prove that for any $\mathbf{g} \in G_n^{m(n)}$ and all sufficiently large $m(n)$, the set $\Lambda(\mathbf{g})$ always contains small nonzero solutions. The main result of this paper (proved in section 6) is that although these small solutions are guaranteed to exist, they are computationally hard to find when $\mathbf{g}$ is chosen uniformly at random.

We know from Proposition 2.11 that $\Lambda(\mathbf{g})$ is a lattice with determinant at most $\det(\Lambda(\mathbf{g})) \leq \#G_n$. We show that $\Lambda(\mathbf{g})$ always contains small solutions by bounding the size of group $G_n$ and then applying Minkowski's first theorem.

Lemma 5.4. *For any $n$, the group $G_n = \mathcal{G}(\mathcal{L}(\mathbf{L}_n), \alpha(n))$ defined in Definition* 5.2 *has size at most*

$$\#G_n \leq \left(\frac{3\alpha(n)\sqrt{n}}{2\lambda_1(\mathcal{L}(\mathbf{L}_n))}\right)^n.$$

*Proof.* The size of the group is $\#G_n = \det(\mathcal{L}(\mathbf{M}_n))/\det(\mathcal{L}(\mathbf{L}_n))$. We bound the two determinants separately. By Corollary 5.3, the columns of $\mathbf{M}_n$ have length at most

$$\|\mathbf{M}_n\mathbf{e}_i\| \leq (3/2)\alpha(n) \cdot \|\mathbf{e}_i\| = 3\alpha(n)/2.$$

Therefore, by Hadamard's inequality

$$\det(\mathcal{L}(\mathbf{M}_n)) \le (3\alpha(n)/2)^n.$$

We bound the determinant of $\mathcal{L}(\mathbf{L}_n)$ using Minkowski's inequality (2.2) $\lambda_1(\mathcal{L}(\mathbf{L}_n)) \le \sqrt{n}\det(\mathcal{L}(\mathbf{L}_n))^{1/n}$. Solving for $\det(\mathcal{L}(\mathbf{L}_n))$, we get that the determinant of $\mathcal{L}(\mathbf{L}_n)$ is at least $(\lambda_1(\mathcal{L}(\mathbf{L}_n))/\sqrt{n})^n$. Combining the two bounds, we get that group $G_n$ has cardinality

$$\#G_n = \frac{\det(\mathcal{L}(\mathbf{M}_n))}{\det(\mathcal{L}(\mathbf{L}_n))} \le \left(\frac{3\alpha(n)\sqrt{n}}{2\lambda_1(\mathcal{L}(\mathbf{L}_n))}\right)^n. \qquad \Box \tag{5.3}$$

A bound on the size of the smallest nonzero solution to equation $\mathbf{g}$ easily follows from Proposition 2.11 and Minkowski's first theorem (2.2).

THEOREM 5.5. *For any equation $\mathbf{g} \in G_n^{m(n)}$ in $m(n) = \Omega(n \log n)$ variables with coefficients in a group $G_n$ of size $\#G_n \le n^{O(n)}$ (e.g., $G_n = \mathcal{G}(\mathcal{L}(\mathbf{L}_n), \alpha(n))$ for some $\alpha(n) = n^{O(1)} \cdot \lambda_1(\mathcal{L}(\mathbf{L}_n))$), there exists a nonzero solution $\mathbf{z} \in \Lambda(\mathbf{g})$ of length at most $\|\mathbf{z}\| = O(\sqrt{m(n)})$.*

## 6. The worst- to average-case reduction.

In this section we prove the main technical result of the paper. Namely, we show that finding short solutions to random linear equations as defined in section 5 (on the average and with nonnegligible probability) is at least as hard as finding linearly independent vectors of length not much bigger than the generalized uniform radius in any lattice (in the worst case and with high probability). Formally, we prove the hardness of the HSIS of Definition 5.1 over groups $\mathcal{G}(\mathcal{L}(\mathbf{L}_n), \alpha(n))$, by reduction from the following variant of SIVP, where the quality of a solution $\|\mathbf{S}\|$, instead of being measured with respect to the size of the smallest possible solution $\lambda_n(\mathcal{L}(\mathbf{B}))$, is measured with respect to some other parameter of interest $\phi(\mathcal{L}(\mathbf{B}))$.

DEFINITION 6.1. *Let $\phi$ be an arbitrary function mapping lattices to positive reals. The* generalized independent vectors problem $\mathrm{GIVP}_\gamma^\phi$*, given a lattice basis $\mathbf{B}$ of rank $n$, asks for a set of $n$ linearly independent lattice vectors $\mathbf{S} \subset \mathcal{L}(\mathbf{B})$ such that $\|\mathbf{S}\| \le \gamma(n) \cdot \phi(\mathcal{L}(\mathbf{B}))$.*

Notice that $\mathrm{SIVP}_\gamma$ is a special case of $\mathrm{GIVP}_\gamma^\phi$, where $\phi = \lambda_n$. Here we are interested in $\mathrm{GIVP}_\gamma^{\hat{\zeta}}$, i.e., the problem of finding a maximal independent set of lattice vectors which are not much longer than the generalized uniform radius.

The reduction is performed in two steps. First we reduce $\mathrm{GIVP}_\gamma^\phi$ to an intermediate problem. Next, we reduce this intermediate problem to the problem of solving random instances of $\mathrm{HSIS}_{G,m,\beta}$. We remark that the intermediate problem is a worst-case one; i.e., the first part of the reduction is a standard worst-case to worst-case Cook reduction. Only the second part of the reduction, from the intermediate problem to the problem of solving random equations, is a worst- to average-case reduction. The advantage of introducing the intermediate problem is that the first part of the reduction (which involves solving many instances of the target problem) is a standard reduction where all problems are solved in the worst case. Once the GIVP problem has been reduced to the intermediate problem, the worst-case to average-case reduction can be expressed in a conceptually simpler setting where a single (worst-case) instance of the intermediate problem is reduced to a single (random) instance of the average-case problem.

The rest of this section is organized as follows. In section 6.1 we reduce GIVP to the intermediate problem and give sufficient conditions for the solutions of the latter.

TABLE 1
*Symbols used in the reduction from* $\mathrm{GIVP}_{\gamma}^{\hat{\zeta}}$ *to* $HSIS_{G,m,\beta}$.

| Symbol | Explanation | Reference |
|---|---|---|
| $\mathbf{L}_n$ | easily decodable $\tau(n)$-perfect lattice | Theorem 6.5 |
| $\mathbf{M}_n$ | almost orthogonal sublattice of $\mathcal{L}(\mathbf{L}_n)$ | Equation (6.2) |
| $\mathbf{B}$ | GIVP input lattice | |
| $\mathbf{C}$ | almost orthogonal sublattice of $\mathcal{L}(\mathbf{B})$ | Equation (6.3) |
| $\mathbf{S}$ | full rank sublattice of $\mathcal{L}(\mathbf{B})$ | |
| $G_n$ | Abelian group ($\mathcal{L}(\mathbf{L}_n)$ modulo $\mathcal{L}(\mathbf{M}_n)$) | Definition 5.1 |
| $\psi$ | Linear function mapping $\mathbf{M}_n$ to $\mathbf{C}$ | Equation (6.4) |
| $(\mathbf{w}_{i,j}, \mathbf{v}_{i,j})$ | vectors in $\mathcal{L}(\mathbf{L}_n) \times \mathcal{L}(\mathbf{B})$ output by the sampling algorithm | Lemma 6.6 |
| $a_{i,j}$ | group element $(\mathbf{w}_{i,j} \bmod \mathbf{M}_n) \in G_n$ | |
| $a_i$ | sum of $a_{i,j}$ for $j = 1, \ldots, k(n)$ | |
| $k(n)$ | Number of samples used to generate each $a_i$ | Equation (6.7) |
| $\mathbf{a}$ | homogeneous linear equation over $G_n$ (input to $\mathcal{F}$) | |
| $\Lambda(\mathbf{a})$ | set of solutions to equation $\mathbf{a}$ | Definition 5.1 |
| $\mathbf{z}$ | solution to equation $\mathbf{a}$ output by $\mathcal{F}$ | |
| $n$ | rank of $\mathbf{L}_n, \mathbf{M}_n, \mathbf{B}, \mathbf{C}$ and $\mathbf{S}$ | |
| $m(n)$ | number of variables in $\mathbf{a}$ | |
| $\alpha(n)$ | scaling factor used in the definition of $\mathbf{M}_n$ | Equation (6.1) |
| $\beta(n)$ | length of the solution $\mathbf{z}$ returned by $\mathcal{F}$ | Theorem 6.5 |
| $\gamma(n)$ | GIVP approximation factor | Theorem 6.5 |
| $\tau(n)$ | upper bound on $2\rho(\mathcal{L}(\mathbf{B}))/\lambda_1(\mathcal{L}(\mathbf{B}))$ | |
| $\mathbf{y}_{i,j}$ | offset vector $\mathbf{v}_{i,j} - \psi(\mathbf{w}_{i,j})$ | |
| $\mathbf{s}$ | output of $\mathcal{A}$ | Equation (6.5) |

In section 6.2 we present the worst- to average-case reduction from the intermediate problem to HSIS (Theorem 6.5). The reduction is based on a sampling procedure (Lemma 6.6) that is described and analyzed in section 6.3. Three technical lemmas (Lemmas 6.8, 6.9, and 6.10) used in the proof of Theorem 6.5 are proved in section 6.4 after establishing some important properties of the sampling procedure. For reference, the notation and symbols used in the reduction are listed in Table 1.

**6.1. The intermediate problem.** In this subsection, we define the intermediate problem, reduce GIVP to it, and present sufficient conditions for its solution. The intermediate problem is essentially an incremental version of GIVP, where given a set of linearly independent vectors $\mathbf{S}$, one has to find a single slightly shorter lattice vector.

DEFINITION 6.2. *The* incremental generalized independent vectors problem ($\mathrm{IncGIVP}_{\gamma}^{\phi}$)*, given a rank n lattice basis* $\mathbf{B}$*, a set of n linearly independent vectors* $\mathbf{S} \subset \mathcal{L}(\mathbf{B})$ *satisfying* $\|\mathbf{S}\| > \gamma(n) \cdot \phi(\mathcal{L}(\mathbf{B}))$*, and an* $(n-1)$*-dimensional hyperplane* $\mathcal{H} \subset \mathrm{span}(\mathbf{B})$*, asks for a lattice vector* $\mathbf{s} \in \mathcal{L}(\mathbf{B}) \setminus \mathcal{H}$ *such that* $\|\mathbf{s}\| \leq \|\mathbf{S}\|/2$.

The following theorem shows that $\mathrm{GIVP}_{\gamma}^{\phi}$ is easily reducible to $\mathrm{IncGIVP}_{\gamma}^{\phi}$.

THEOREM 6.3. *For any functions* $\phi$ *and* $\gamma$*, there is a polynomial time (Cook) reduction from* $\mathrm{GIVP}_{\gamma}^{\phi}$ *to* $\mathrm{IncGIVP}_{\gamma}^{\phi}$.

*Proof.* We give an iterative reduction (see section 2.6) from $\mathrm{GIVP}_{\gamma}^{\phi}$ to $\mathrm{IncGIVP}_{\gamma}^{\phi}$. By Theorem 2.21, this immediately implies a standard Cook reduction between the two problems. Let $\mathcal{A}(\mathbf{B}, \mathbf{S}, \mathcal{H})$ be an algorithm solving $\mathrm{IncGIVP}_{\gamma}^{\phi}$ in the worst case. The iterative reduction $(R, f, I, O, \mathcal{S})$ is defined as follows. Relation $R$ is the set of all $(\mathbf{B}, \mathbf{S})$ where $\mathbf{B}$ is the GIVP input lattice, and $\mathbf{S} \subset \mathcal{L}(\mathbf{B})$ is a maximal set of linearly independent lattice vectors such that $\|\mathbf{S}\| \leq \|\mathbf{B}\|$. (This condition is introduced to make sure that the size of $\mathbf{S}$ is polynomial in the input size.) Initially, $\mathbf{S}$ is set to the input basis $I(\mathbf{B}) = \mathbf{B}$. Upon termination, the iterative reduction outputs the current set $O(\mathbf{B}, \mathbf{S}) = \mathbf{S}$. Progress at each iteration is measured by the function $f(\mathbf{B}, \mathbf{S}) = \prod_{i=1}^{n} \|\mathbf{s}_i\|^2$. Notice that function $f$ is polynomial time computable. In order to complete the iterative reduction we give a polynomial time oracle algorithm $\mathcal{S}^{\mathcal{A}}$ (the iterative step) that on input a rank $n$ lattice basis $\mathbf{B}$ and $n$ linearly independent lattice vectors $\mathbf{S} \subset \mathcal{L}(\mathbf{B})$ such that $\gamma(n) \cdot \phi(\mathcal{L}(\mathbf{B})) < \|\mathbf{S}\| \leq \|\mathbf{B}\|$, finds a set of linearly independent lattice vectors $\mathbf{S}'$ such that $\|\mathbf{S}'\| \leq \|\mathbf{B}\|$ and $f(\mathbf{S}') \leq f(\mathbf{S})/2$.

Algorithm $\mathcal{S}^{\mathcal{A}}(\mathbf{B}, \mathbf{S})$ works as follows. Let $i$ be the index of a longest vector in $\mathbf{S}$, i.e., $\|\mathbf{s}_i\| = \|\mathbf{S}\|$, and let $\mathcal{H} = \operatorname{span}(\mathbf{s}_1, \ldots, \mathbf{s}_{i-1}, \mathbf{s}_{i+1}, \ldots, \mathbf{s}_n)$ be the $(n-1)$-dimensional hyperplane spanned by the other vectors. The iterative step $\mathcal{S}$ computes $\mathbf{s} = \mathcal{A}(\mathbf{B}, \mathbf{S}, \mathcal{H})$ and checks that vector $\mathbf{s}$ satisfies $\mathbf{s} \in \mathcal{L}(\mathbf{B}) \setminus \mathcal{H}$ and $\|\mathbf{s}\| \leq \|\mathbf{S}\|/2$. If so, then $\mathcal{S}$ replaces $\mathbf{s}_i$ with $\mathbf{s}$ and outputs $\mathbf{S}' = [\mathbf{s}_1, \ldots, \mathbf{s}_{i-1}, \mathbf{s}, \mathbf{s}_{i+1}, \ldots, \mathbf{s}_n]$. Otherwise, $\mathcal{S}$ simply outputs the input set $\mathbf{S}' = \mathbf{S}$. Notice that in both cases, the output $\mathbf{S}'$ satisfies the relation $(\mathbf{B}, \mathbf{S}') \in R$; i.e., $\mathbf{S}' \subset \mathcal{L}(\mathbf{B})$ is a set of $n$ linearly independent lattice vectors and $\|\mathbf{S}'\| \leq \|\mathbf{S}\| \leq \|\mathbf{B}\|$. Moreover, if $\|\mathbf{S}\| > \gamma(n) \cdot \phi(\mathcal{L}(\mathbf{B}))$, then $\mathcal{A}(\mathbf{B}, \mathbf{S}, \mathcal{H})$ successfully returns a vector $\mathbf{s} \in \mathcal{L}(\mathbf{B}) \setminus \mathcal{H}$ satisfying $\|\mathbf{s}\| \leq \|\mathbf{S}\|/2$, and

$$f(\mathbf{B}, \mathbf{S}') = f(\mathbf{B}, \mathbf{S}) \frac{\|\mathbf{s}\|^2}{\|\mathbf{s}_i\|^2} = f(\mathbf{B}, \mathbf{S}) \frac{\|\mathbf{s}\|^2}{\|\mathbf{S}\|^2} \leq \frac{f(\mathbf{B}, \mathbf{S})}{4}. \qquad \square$$

The following lemma establishes sufficient conditions for reducing IncGIVP (in the worst case) to HSIS (on the average).

Lemma 6.4. *Let $\mathcal{A}^{(\cdot)}(\mathbf{B}, \mathbf{S}, \mathcal{H})$ be a probabilistic polynomial time oracle algorithm that on input a rank $n$ lattice basis $\mathbf{B}$, a full rank subset $\mathbf{S} \subset \mathcal{L}(\mathbf{B})$, and an $(n-1)$-dimensional hyperplane $\mathcal{H} \subset \operatorname{span}(\mathbf{B})$, makes a single oracle call $\mathbf{a} \in G_n^{m(n)}$ and (provided the query is answered with a valid solution $\mathbf{z} \in \Lambda(\mathbf{a})$) outputs a lattice vector $\mathbf{s} \in \mathcal{L}(\mathbf{B})$. Assume that for any input $(\mathbf{B}, \mathbf{S}, \mathcal{H})$ such that $\|\mathbf{S}\| > \gamma(n) \cdot \phi(\mathcal{L}(\mathbf{B}))$, the vectors $\mathbf{a}, \mathbf{z}, \mathbf{s}$ produced by $\mathcal{A}^{(\cdot)}(\mathbf{B}, \mathbf{S}, \mathcal{H})$ satisfy the following properties:*

(i) *the statistical distance between the query $\mathbf{a}$ and a uniformly distributed $\mathbf{u} \in G_n^{m(n)}$ is negligible, i.e.,*

$$\Delta(\mathbf{a}, \mathbf{u}) = n^{-\omega(1)},$$

(ii) *for any $\hat{\mathbf{a}} \in G_n^{m(n)}$ and $\hat{\mathbf{z}} \in \Lambda(\hat{\mathbf{a}}) \setminus \{\mathbf{0}\}$, the conditional probability that $\mathbf{s} \notin \mathcal{H}$ is at least*

$$\Pr\{\mathbf{s} \notin \mathcal{H} \mid \mathbf{a} = \hat{\mathbf{a}}, \mathbf{z} = \hat{\mathbf{z}}\} = \Omega(1),$$

(iii) *for any $\hat{\mathbf{a}} \in G_n^{m(n)}$ and $\hat{\mathbf{z}} \in \Lambda(\hat{\mathbf{a}})$, the conditional expectation of $\|\mathbf{s}\|$ is at most*

$$\operatorname{Exp}[\|\mathbf{s}\| \mid \mathbf{a} = \hat{\mathbf{a}}, \mathbf{z} = \hat{\mathbf{z}}] = o\left(\frac{\|\hat{\mathbf{z}}\| \cdot \|\mathbf{S}\|}{\beta(n)}\right).$$

*Then, for any randomized procedure $\mathcal{F}$ that solves $\mathrm{HSIS}_{G,m,\beta}$ on the average with nonnegligible probability $\delta(n)$, $\mathcal{A}^{\mathcal{F}}(\mathbf{B}, \mathbf{S}, \mathcal{H})$ solves $\mathrm{IncGIVP}_{\gamma}^{\phi}$ in the worst case with high probability*[16] *$\Omega(\delta(n))$.*

*Proof.* Let $\mathcal{F}$ be a randomized procedure that solves $\mathrm{HSIS}_{G,m,\beta}$ with nonnegligible probability $\delta(n)$. We want to prove that, for any valid input, $\mathcal{A}^{\mathcal{F}}(\mathbf{B}, \mathbf{S}, \mathcal{H})$ solves

[16]Remember that, since $\mathcal{A}$ solves IncGIVP in the *worst case*, and given a vector $\mathbf{s}$ it is easy to check if $\mathbf{s}$ is a correct solution to an IncGIVP instance, the success probability of $\mathcal{A}$ can be efficiently boosted from any nonnegligible fraction to exponentially close to one.

$\text{IncGIVP}_\gamma^\phi$ with probability $\Omega(\delta(n))$. Namely, we want to prove that for any rank $n$ lattice basis $\mathbf{B}$, full rank subset $\mathbf{S} \subset \mathcal{L}(\mathbf{B})$ such that $\|\mathbf{S}\| > \gamma(n) \cdot \phi(\mathcal{L}(\mathbf{B}))$, and $(n-1)$-dimensional hyperplane $\mathcal{H} \subset \operatorname{span}(\mathbf{B})$, procedure $\mathcal{A}^{\mathcal{F}}(\mathbf{B}, \mathbf{S}, \mathcal{H})$ outputs a lattice vector $\mathbf{s} \in \mathcal{L}(\mathbf{B}) \setminus \mathcal{H}$ of length $\|\mathbf{s}\| \le \|\mathbf{S}\|/2$ with nonnegligible probability $\Omega(\delta(n))$.

Assume without loss of generality that $\mathcal{F}(\mathbf{a})$ always returns a (possibly zero) solution $\mathcal{F}(\mathbf{a}) \in \Lambda(\mathbf{a})$ of length $\|\mathcal{F}(\mathbf{a})\| \le \beta(n)$. The assumption on $\mathcal{F}$ is that $\Pr\{\mathcal{F}(\mathbf{u}) \neq \mathbf{0}\} = \delta(n)$ when $\mathbf{u} \in G_n^{m(n)}$ is chosen uniformly at random. Since $\mathcal{F}(\mathbf{a})$ always returns a valid solution $\mathbf{z} \in \Lambda(\mathbf{a})$, the output vector $\mathbf{s}$ is guaranteed to belong to the lattice $\mathcal{L}(\mathbf{B})$. We need to bound the probability that $\mathbf{s}$ also satisfies $\mathbf{s} \notin \mathcal{H}$ and $\|\mathbf{s}\| \le \|\mathbf{S}\|/2$. Consider an execution of $\mathcal{A}^{\mathcal{F}}(\mathbf{B}, \mathbf{S}, \mathcal{H}) = \mathbf{s}$, and let $\mathcal{F}(\mathbf{a}) = \mathbf{z}$ be the oracle call made by $\mathcal{A}$. Conditioning on the value of $\mathbf{a}$ and $\mathbf{z}$, and restricting our attention to the nonzero solutions $\mathbf{z} \neq \mathbf{0}$, we get

$$\begin{aligned}
&\Pr\{\mathbf{s} \notin \mathcal{H} \wedge \|\mathbf{s}\| \le \|\mathbf{S}\|/2\} \\
&\qquad = \sum_{\hat{\mathbf{a}}, \hat{\mathbf{z}}} \Pr\{\mathbf{a} = \hat{\mathbf{a}} \wedge \mathbf{z} = \hat{\mathbf{z}}\} \cdot \Pr\{\mathbf{s} \notin \mathcal{H} \wedge \|\mathbf{s}\| \le \|\mathbf{S}\|/2 \mid \mathbf{a} = \hat{\mathbf{a}} \wedge \mathbf{z} = \hat{\mathbf{z}}\} \\
&\qquad \ge \sum_{\hat{\mathbf{a}}, \hat{\mathbf{z}} : \hat{\mathbf{z}} \neq \mathbf{0}} \Pr\{\mathbf{a} = \hat{\mathbf{a}} \wedge \mathbf{z} = \hat{\mathbf{z}}\} \\
&\qquad\qquad \cdot \left(\Pr\{\mathbf{s} \notin \mathcal{H} \mid \mathbf{a} = \hat{\mathbf{a}} \wedge \mathbf{z} = \hat{\mathbf{z}}\} - \Pr\{\|\mathbf{s}\| > \|\mathbf{S}\|/2 \mid \mathbf{a} = \hat{\mathbf{a}} \wedge \mathbf{z} = \hat{\mathbf{z}}\}\right),
\end{aligned}$$

where the summations range over all $\hat{\mathbf{a}} \in G_n^{m(n)}$ and $\hat{\mathbf{z}} \in [\mathcal{F}(\hat{\mathbf{a}})] \subseteq \Lambda(\hat{\mathbf{a}}) \cap \mathcal{B}(\beta(n))$. By assumption on $\mathcal{A}$, for any $\hat{\mathbf{a}} \in G_n^{m(n)}$ and $\hat{\mathbf{z}} \in \Lambda(\hat{\mathbf{a}})$ such that $0 < \|\hat{\mathbf{z}}\| \le \beta(n)$, the two conditional probabilities in the last expression satisfy

$$\Pr\{\mathbf{s} \notin \mathcal{H} \mid \mathbf{a} = \hat{\mathbf{a}} \wedge \mathbf{z} = \hat{\mathbf{z}}\} = \Omega(1),$$

and, using Markov's inequality,

$$\begin{aligned}
\Pr\{\|\mathbf{s}\| > \|\mathbf{S}\|/2 \mid \mathbf{a} = \hat{\mathbf{a}} \wedge \mathbf{z} = \hat{\mathbf{z}}\} &\le \frac{\operatorname{Exp}[\|\mathbf{s}\| \mid \mathbf{a} = \hat{\mathbf{a}} \wedge \mathbf{z} = \hat{\mathbf{z}}]}{\|\mathbf{S}\|/2} \\
&\le o\left(\frac{2\|\hat{\mathbf{z}}\| \cdot \|\mathbf{S}\|}{\|\mathbf{S}\| \cdot \beta(n)}\right) = o(1).
\end{aligned}$$

Adding up for all possible values of $\hat{\mathbf{a}}$ and $\hat{\mathbf{z}} \neq \mathbf{0}$, we get

$$\begin{aligned}
\Pr\{\mathbf{s} \notin \mathcal{H} \wedge \|\mathbf{s}\| \le \|\mathbf{S}\|/2\} &\ge \sum_{\hat{\mathbf{a}}, \hat{\mathbf{z}} : \hat{\mathbf{z}} \neq \mathbf{0}} \Pr\{\mathbf{a} = \hat{\mathbf{a}} \wedge \mathbf{z} = \hat{\mathbf{z}}\} \cdot (\Omega(1) - o(1)) \\
&= \Omega(\Pr\{\mathbf{z} \neq \mathbf{0}\}).
\end{aligned}$$

Notice that $\mathbf{z} = \mathcal{F}(\mathbf{a})$ and $\Pr\{\mathcal{F}(\mathbf{u}) \neq \mathbf{0}\} = \delta(n)$ when $\mathbf{u} \in G_n^{m(n)}$ is uniformly distributed. By assumption, the statistical distance $\Delta(\mathbf{a}, \mathbf{u})$ between $\mathbf{a}$ and $\mathbf{u}$ is negligible. Therefore, by Corollary 2.16,

$$\begin{aligned}
\Pr\{\mathbf{z} \neq \mathbf{0}\} &= \Pr\{\mathcal{F}(\mathbf{a}) \neq \mathbf{0}\} \\
&\ge \Pr\{\mathcal{F}(\mathbf{u}) \neq \mathbf{0}\} - \Delta(\mathbf{a}, \mathbf{u}) \\
&\ge \delta(n) - n^{-\omega(1)}.
\end{aligned}$$

So, for all nonnegligible $\delta(n)$, $\Pr\{\mathbf{s} \notin \mathcal{H} \wedge \|\mathbf{s}\| \le \|\mathbf{S}\|/2\} \ge \Omega(\delta(n) - n^{-\omega(1)}) = \Omega(\delta(n))$. $\square$

**6.2. The main reduction.** In this subsection we show that for appropriate choice of groups $G_n$, and parameters $\beta(n), m(n), \gamma(n)$, there is a reduction from solving $\text{IncGIVP}_\gamma^{\hat{\zeta}}$ in the worst case to solving $\text{HSIS}_{G,m,\beta}$ on the average.

THEOREM 6.5. *Let $\tau(n) \geq 1$ such that there exists an easily decodable family of $\tau(n)$-perfect lattices. Then, for any $\beta(n) \geq 1$, $m(n) = n^{O(1)}$ and $\gamma(n) = \beta(n)\tau(n) \cdot \sqrt{\omega(\log n)}$, there is a sequence of efficiently computable Abelian groups $G_n$ of size $\#G_n \leq (n^{1.5}\gamma(n)/8)^n$ such that solving $\text{IncGIVP}_\gamma^{\hat{\zeta}}$ in the worst case with high probability reduces to solving $\text{HSIS}_{G,m,\beta}$ on the average with nonnegligible probability.*

*Proof.* Let $\{\mathcal{L}(\mathbf{L}_n)\}$ be a family of easily decodable $\tau(n)$-perfect lattices. For any $\beta(n) \geq 1$ and $m(n) = n^{O(1)}$, let $\gamma(n) = \beta(n)\tau(n) \cdot \sqrt{\omega(\log n)}$ and

$$\alpha(n) = \frac{n\lambda_1(\mathcal{L}(\mathbf{L}_n))\gamma(n)}{12}. \tag{6.1}$$

Notice that from the definition of $\alpha(n)$ and $\gamma(n)$, and the assumption that $\mathcal{L}(\mathbf{L}_n)$ is $\tau(n)$-perfect, we get

$$\begin{aligned}\alpha(n) &= \frac{n\lambda_1(\mathcal{L}(\mathbf{L}_n))\beta(n)\tau(n)\sqrt{\omega(\log n)}}{12} \\ &\geq \left(\frac{\beta(n)\sqrt{n \cdot \omega(\log n)}}{12}\right) 2\sqrt{n}\rho(\mathcal{L}(\mathbf{L}_n)) \\ &\geq 2\sqrt{n}\rho(\mathcal{L}(\mathbf{L}_n)).\end{aligned}$$

So, $\alpha(n)$ satisfies the condition in Definition 5.2, and we can define a full rank subset $\mathbf{M}_n \subseteq \mathcal{L}(\mathbf{L}_n)$ and quotient group $G_n = \mathcal{G}(\mathcal{L}(\mathbf{L}_n), \alpha(n)) = \mathcal{L}(\mathbf{L}_n)/\mathcal{L}(\mathbf{M}_n)$ such that Corollary 5.3 and Lemma 5.4 hold true; i.e.,

$$\forall \mathbf{x} \in \mathbb{R}^n. \|\mathbf{M}_n\mathbf{x}\| \approx \alpha(n) \cdot \|\mathbf{x}\| \tag{6.2}$$

and group $G_n$ has size at most

$$\#G_n \leq \left(\frac{3\alpha(n)\sqrt{n}}{2\lambda_1(\mathcal{L}(\mathbf{L}_n))}\right)^n = \left(\frac{n^{1.5}\gamma(n)}{8}\right)^n.$$

We define a probabilistic polynomial time oracle algorithm $\mathcal{A}^{(\cdot)}$ satisfying the conditions in Lemma 6.4 with $\phi = \hat{\zeta}$. It follows from Lemma 6.4 that $\mathcal{A}^{(\cdot)}$ is a probabilistic polynomial time worst-case to average-case reduction from $\text{IncGIVP}_\gamma^{\hat{\zeta}}$ to $\text{HSIS}_{G,m,\beta}$. The intuition behind procedure $\mathcal{A}^{(\cdot)}$ is the following. (See Figure 2.) Map $\mathcal{L}(\mathbf{M}_n)$ to a sublattice $\mathcal{L}(\mathbf{C}) = \psi(\mathcal{L}(\mathbf{M}_n)) \subset \mathcal{L}(\mathbf{B})$ using a linear function $\psi$ with small distortion, i.e., a function that approximately preserves distance ratios. One possible way to achieve this is to map the almost orthogonal set $\mathbf{M}_n$ to an almost orthogonal subset $\mathbf{C} = \psi(\mathbf{M}_n) \subset \mathcal{L}(\mathbf{B})$ and extend $\psi$ to span$(\mathbf{M}_n)$ by linearity. Now, consider the Voronoi cells $\mathcal{V}(\mathbf{w}, \mathcal{L}(\mathbf{L}_n))$ of the $\tau(n)$-perfect lattice $\mathcal{L}(\mathbf{L}_n)$. Function $\psi$ maps each cell to a corresponding region $\psi(\mathcal{V}(\mathbf{w}, \mathcal{L}(\mathbf{L}_n)))$ centered around $\psi(\mathbf{w})$. Partition the points of $\mathcal{L}(\mathbf{B})$ into subsets, according to these regions. Pick $m(n)$ points $\mathbf{v}_i \in \mathcal{L}(\mathbf{B})$ at random, and map each of them to the center $\psi(\mathbf{w}_i)$ of the corresponding region. Notice that each region $\psi(\mathcal{V}(\mathbf{w}, \mathcal{L}(\mathbf{L}_n)))$ is associated to a group element $[\mathbf{w}]_{\mathbf{M}_n} \in G_n$. So, the points $\mathbf{v}_i$ define $m(n)$ group elements $a_i = [\mathbf{w}_i]_{\mathbf{M}_n} \in G_n$. Use $\mathcal{F}$ to find a small nonzero solution $\mathbf{z} = \mathcal{F}(\mathbf{a})$ to the equation $\mathbf{a} = [a_1, \ldots, a_{m(n)}]$. The output of $\mathcal{A}^{\mathcal{F}}(\mathbf{B}, \mathbf{S}, \mathcal{H})$ is vector $\mathbf{s} = \sum_i z_i(\mathbf{v}_i - \psi(\mathbf{w}_i))$. Notice that $\mathbf{s} \in \mathcal{L}(\mathbf{B})$ because $\sum z_i\mathbf{v}_i$ is

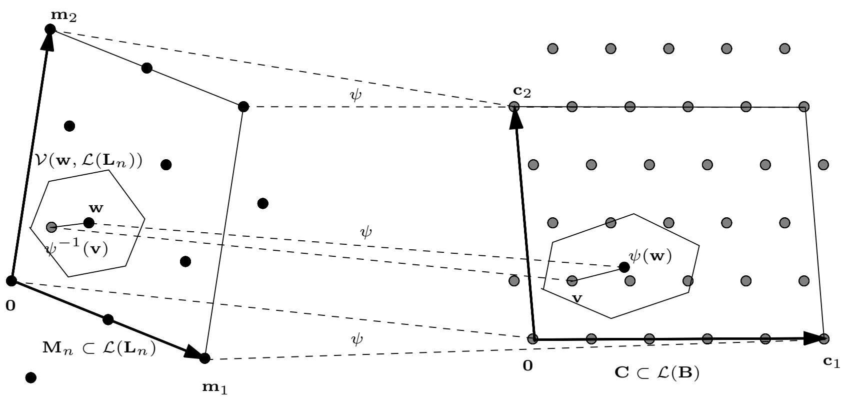


FIG. 2. *Sampling lattice points.*

an integer combination of lattice vectors, and $\sum_i z_i \mathbf{w}_i \in \mathcal{L}(\mathbf{M}_n) \subseteq \psi^{-1}(\mathcal{L}(\mathbf{B}))$. (See Lemma 6.7 for details.) Before moving to the actual proof, we informally explain why vectors $\mathbf{a}, \mathbf{z}, \mathbf{s}$ are expected to satisfy the three conditions in Lemma 6.4. (1) Vector $\mathbf{a}$ is distributed almost uniformly at random because coefficients $a_i = [\mathbf{w}_i]_{\mathbf{M}_n}$ are chosen independently, and each region $\psi(\mathcal{V}(\mathbf{w}, \mathcal{L}(\mathbf{L}_n)))$ contains roughly the same number of lattice points from $\mathcal{L}(\mathbf{B})$. (2) Vector $\mathbf{s}$ does not belong to any fixed hyperplane $\mathcal{H}$ with high probability because each $\mathbf{v}_i - \psi(\mathbf{w}_i)$ is somehow randomly distributed within $\psi(\mathcal{V}(\mathcal{L}(\mathbf{L}_n)))$. (3) Finally, $\mathbf{s}$ is short because it is a small combination of short vectors $\mathbf{v}_i - \psi(\mathbf{w}_i)$, each one lying within the region $\psi(\mathcal{V}(\mathcal{L}(\mathbf{L}_n)))$. This is an oversimplified description of the reduction. For example, Lemma 6.4 requires distribution $\mathbf{a}$ to be extremely close to uniform. In order to ensure the almost uniform distribution of $\mathbf{a}$, we will need to slightly modify the above procedure by sampling many points $(\mathbf{w}_{i,j}, \mathbf{v}_{i,j})$ and adding up the corresponding $a_{i,j} = [\mathbf{w}_{i,j}]_{\mathbf{M}_n}$ to obtain group elements $a_i = \sum_j a_{i,j}$ whose distribution is extremely close to uniform.

We now give a detailed description of procedure $\mathcal{A}^{\mathcal{F}}(\mathbf{B}, \mathbf{S}, \mathcal{H})$. Notice that the procedure outlined above does not use the input hyperplane $\mathcal{H}$, and condition $\mathbf{s} \notin \mathcal{H}$ holds with high probability for any fixed hyperplane $\mathcal{H}$. Therefore, below we simply write $\mathcal{A}^{\mathcal{F}}(\mathbf{B}, \mathbf{S})$ instead of $\mathcal{A}^{\mathcal{F}}(\mathbf{B}, \mathbf{S}, \mathcal{H})$ to emphasize the fact that $\mathcal{A}$ does not use the input hyperplane $\mathcal{H}$.

Procedure $\mathcal{A}^{(\cdot)}(\mathbf{B}, \mathbf{S})$ works as follows. First, notice that using Babai's nearest plane algorithm [6], matrix $\mathbf{S}$ allows us to approximate any vector $\mathbf{x} \in \operatorname{span}(\mathbf{B})$ with a lattice point $\mathbf{y} \in \mathcal{L}(\mathbf{S}) \subseteq \mathcal{L}(\mathbf{B})$ within distance $\sigma = (\sqrt{n}/2)\|\mathbf{S}\|$ from $\mathbf{x}$.[17] Therefore, using Lemma 2.10, we can find an almost orthogonal sublattice $\mathcal{L}(\mathbf{C}) \subset \mathcal{L}(\mathbf{B})$ such that

$$\forall \mathbf{x} \in \mathbb{R}^n . \|\mathbf{C}\mathbf{x}\| \approx n\|\mathbf{S}\| \cdot \|\mathbf{x}\|. \tag{6.3}$$

Let $\psi(\mathbf{x}) = \mathbf{C}\mathbf{M}_n^{-1}\mathbf{x}$ be the linear transformation that maps $\mathbf{m}_i$ to $\mathbf{c}_i$ for all $i = 1, \ldots, n$. Combining (6.2) and (6.3), and using (2.1), we get

$$\forall \mathbf{x} \in \mathbb{R}^n . \frac{1}{3}\left(\frac{n\|\mathbf{S}\|}{\alpha(n)}\right) \cdot \|\mathbf{x}\| \le \|\psi(\mathbf{x})\| \le 3\left(\frac{n\|\mathbf{S}\|}{\alpha(n)}\right) \cdot \|\mathbf{x}\|; \tag{6.4}$$

[17]This is not a particularly critical part of the reduction, and using poorer rounding procedures (e.g., rounding off the coordinates of $\mathbf{x}$ with respect to basis $\mathbf{S}$ to the closest integers as done in [2]) results in substantially the same connection factors as using Babai's nearest plane algorithm [6].

i.e., the linear function $\psi$ is close to an orthogonal transformation that scales all distances by a factor $n\|\mathbf{S}\|/\alpha(n)$.

Notice that $\mathcal{L}(\mathbf{M}_n)$ is a common sublattice of both $\mathcal{L}(\mathbf{L}_n)$ and $\psi^{-1}(\mathcal{L}(\mathbf{B}))$. The following lemma shows how to use function $\psi$ together with decoding algorithm $\mathrm{CVP}_{\mathbf{L}}$ to simultaneously sample from groups $G_n = \mathcal{L}(\mathbf{L}_n)/\mathcal{L}(\mathbf{M}_n)$ and $\mathcal{L}(\mathbf{B})/\mathcal{L}(\mathbf{C}) \equiv \psi^{-1}(\mathcal{L}(\mathbf{B}))/\mathcal{L}(\mathbf{M}_n)$.

LEMMA 6.6. *There is a sampling algorithm that on input two rank $n$ lattices $\mathbf{L}_n$ and $\mathbf{B}$, a full rank sublattice $\mathbf{M}_n \subset \mathcal{L}(\mathbf{L}_n)$, and a nonsingular linear transformation $\psi$ such that $\mathbf{C} = \psi(\mathbf{M}_n) \subset \mathcal{L}(\mathbf{B})$, outputs two vectors $\mathbf{w} \in \mathcal{L}(\mathbf{L}_n)$ and $\mathbf{v} \in \mathcal{L}(\mathbf{B})$ such that the following hold:*

1. *The group element $[\mathbf{v}]_{\mathbf{C}}$ is uniformly distributed over $\mathcal{L}(\mathbf{B})/\mathcal{L}(\mathbf{C})$.*
2. *$\psi^{-1}(\mathbf{v}) \in \bar{\mathcal{V}}(\mathbf{w}, \mathcal{L}(\mathbf{L}_n))$, or, equivalently, $\mathbf{v} - \psi(\mathbf{w}) \in \psi(\bar{\mathcal{V}}(\mathcal{L}(\mathbf{L}_n)))$.*
3. *The distribution of $\mathbf{v} - \psi(\mathbf{w})$ is symmetric about the origin, and, in particular, $\operatorname{Exp}[\mathbf{v} - \psi(\mathbf{w})] = 0$.*
4. *$\mathbf{w} \in \mathcal{P}(\mathbf{M}_n)$.*

*Moreover, if lattice $\mathbf{L}_n$ is easily decodable, then the sampling procedure runs in polynomial time.*

The actual properties of the sampling algorithm are not important at this point, and the proof of Lemma 6.6 is deferred to section 6.3. All that matters for now is that the sampling algorithm generates pairs of vectors $(\mathbf{w}, \mathbf{v}) \in \mathcal{L}(\mathbf{L}_n) \times \mathcal{L}(\mathbf{B})$. Below we describe how to use any such sampling procedure to compute a lattice vector $\mathbf{s} \in \mathcal{L}(\mathbf{B})$. After defining the full rank sublattice $\mathbf{C} \subset \mathcal{L}(\mathbf{S})$ and linear function $\psi(\mathbf{M}_n) = \mathbf{C}$ satisfying (6.4), algorithm $\mathcal{A}^{\mathcal{F}}(\mathbf{B}, \mathbf{S})$ proceeds as follows:

1. Run the sampling procedure of Lemma 6.6 $m(n) \cdot k(n)$ times (where $k(n) = \omega(\log n)$ is a superlogarithmic function to be specified) to generate vectors $\mathbf{w}_{i,j} \in \mathcal{L}(\mathbf{L}_n)$ and $\mathbf{v}_{i,j} \in \mathcal{L}(\mathbf{B})$ for $i = 1, \ldots, m(n)$ and $j = 1, \ldots, k(n)$.
2. Let $a_{i,j} = [\mathbf{w}_{i,j}]_{\mathbf{M}_n} \in G_n$ be the group elements corresponding to lattice points $\mathbf{w}_{i,j}$ and, for every $i = 1, \ldots, m(n)$, define group element $a_i = \sum_{j=1}^{k(n)} a_{i,j}$.
3. Use oracle $\mathcal{F}$ to compute a nonzero solution $\mathbf{z} = \mathcal{F}(\mathbf{a}) \in \Lambda(\mathbf{a}) \setminus \{\mathbf{0}\}$ to the equation $\mathbf{a} = [a_1, \ldots, a_{m(n)}]$.
4. For any $i, j$, let $\mathbf{y}_{i,j} = \mathbf{v}_{i,j} - \psi(\mathbf{w}_{i,j})$, and output

$$\mathbf{s} = \sum_{i=1}^{m(n)} z_i \sum_{j=1}^{k(n)} \mathbf{y}_{i,j}. \tag{6.5}$$

Notice that randomness is used twice in the routine: first in step 1 and then in step 3. In step 1, randomness is used to run the sampling procedure $m(n) \cdot k(n)$ times and generate a random equation $\mathbf{a}$ to be passed as input to $\mathcal{F}$. In step 3 randomness is used to run the probabilistic procedure $\mathcal{F}$ on input $\mathbf{a}$ to compute a solution $\mathbf{z}$. Since $\mathcal{F}$ is only guaranteed to work *on the average*, it is important that both the input $\mathbf{a}$ and the internal randomness of $\mathcal{F}$ are chosen (almost) uniformly and independently at random. We remark that, although the value of $\mathbf{z}$ depends on both the randomness used by the sampling procedure and that used directly by $\mathcal{F}$, the two procedures use independent sources of randomness. So, for example, given the value of $\mathbf{a}$, the conditional distribution of $\mathbf{z}$ is independent from the conditional distribution of the samples $(\mathbf{w}_{i,j}, \mathbf{v}_{i,j})$. We will use this fact in the probabilistic analysis of the success probability of the reduction.

In the following lemma we prove that algorithm $\mathcal{A}^{\mathcal{F}}$ is correct, i.e., the output vector $\mathbf{s}$ belongs to lattice $\mathcal{L}(\mathbf{B})$, provided query $\mathbf{a}$ is answered with a valid solution $\mathbf{z} \in \Lambda(\mathbf{a})$.

Lemma 6.7. *Let* $\mathbf{s}$ *be the output vector defined in* (6.5). *If* $\mathbf{z} \in \Lambda(\mathbf{a})$, *then* $\mathbf{s} \in \mathcal{L}(\mathbf{B})$.

*Proof.* Define the vector

$$\mathbf{w} = \sum_{i=1}^{m(n)} z_i \sum_{j=1}^{k(n)} \mathbf{w}_{i,j}.$$

Using the definition of $\mathbf{y}_{i,j}$ and the linearity of $\psi$, we get

$$\mathbf{s} = \sum_{i=1}^{m(n)} z_i \sum_{j=1}^{k(n)} \mathbf{y}_{i,j} = \sum_{i,j} z_i(\mathbf{v}_{i,j} - \psi(\mathbf{w}_{i,j})) = \left(\sum_{i,j} z_i \mathbf{v}_{i,j}\right) - \psi(\mathbf{w}).$$

The first term $\sum_{i,j} z_i \mathbf{v}_{i,j}$ clearly belongs to $\mathcal{L}(\mathbf{B})$ because it is an integer linear combination of lattice vectors $\mathbf{v}_{i,j} \in \mathcal{L}(\mathbf{B})$. We need to prove that the second term $\psi(\mathbf{w})$ also belongs to $\mathcal{L}(\mathbf{B})$. We show that $\mathbf{w} \in \mathcal{L}(\mathbf{M}_n)$. Since $\psi$ maps $\mathcal{L}(\mathbf{M}_n)$ to $\mathcal{L}(\mathbf{C})$, it follows that $\psi(\mathbf{w}) \in \mathcal{L}(\mathbf{C}) \subseteq \mathcal{L}(\mathbf{B})$.

Remember that $\mathbf{z} = \mathcal{F}(\mathbf{a}) \in \Lambda(\mathbf{a})$, i.e., $\sum_i z_i a_i = 0$ (in $G_n$). Since all $\mathbf{w}_{i,j}$ belong to $\mathcal{L}(\mathbf{L}_n)$, $\mathbf{w}$ is also a lattice point of $\mathcal{L}(\mathbf{L}_n)$ and $[\mathbf{w}]_{\mathbf{M}_n} \in G_n$. The group element corresponding to lattice vector $\mathbf{w}$ is

$$[\mathbf{w}]_{\mathbf{M}_n} = \sum_{i=1}^{m(n)} z_i \sum_{j=1}^{k(n)} [\mathbf{w}_{i,j}]_{\mathbf{M}_n} = \sum_i z_i \sum_j a_{i,j} = \sum_i z_i a_i = 0.$$

Since $G_n$ is the quotient of $\mathcal{L}(\mathbf{L}_n)$ modulo $\mathcal{L}(\mathbf{M}_n)$, this proves that $\mathbf{w} \in \mathcal{L}(\mathbf{M}_n)$. ◻

The following three lemmas show that, provided $\alpha(n)$ is in a prescribed range, procedure $\mathcal{A}$ satisfies the conditions in Lemma 6.4. The lemmas are proved in section 6.4, after establishing some useful properties of the sampling procedure in section 6.3.

The first lemma shows that the equation $\mathbf{a}$ passed as input to oracle $\mathcal{F}$ is almost uniformly distributed.

Lemma 6.8. *If* $n\|\mathbf{S}\|\lambda_1(\mathcal{L}(\mathbf{L}_n)) \geq 6\alpha(n)\hat{\zeta}(\mathcal{L}(\mathbf{B}))$ *and* (6.4) *holds true, then the statistical distance between vector* $\mathbf{a}$ *(passed as input to* $\mathcal{F}$ *during the execution of* $\mathcal{A}^{\mathcal{F}}(\mathbf{B},\mathbf{S})$*) and a uniformly distributed* $\mathbf{u} \in G_n^{m(n)}$ *is at most* $\Delta(\mathbf{a},\mathbf{u}) \leq m(n)/2^{k(n)+1}$. *In particular, for any polynomially bounded* $m(n) = n^{O(1)}$ *and superlogarithmic function* $k(n) = \omega(\log n)$, *the statistical distance* $\Delta(\mathbf{a},\mathbf{u}) = n^{-\omega(1)}$ *is negligible.*

The other two lemmas show that the output vector $\mathbf{s}$ of procedure $\mathcal{A}^{\mathcal{F}}$ is sufficiently random and usually short, even after conditioning on the input and output values of oracle $\mathcal{F}$.

Lemma 6.9. *Assume* $n\|\mathbf{S}\|\lambda_1(\mathcal{L}(\mathbf{L}_n)) \geq 12\alpha(n)\hat{\zeta}(\mathcal{L}(\mathbf{B}))$ *and* (6.4) *holds true. Then, for any* $\hat{\mathbf{a}} \in G_n^{m(n)}$, $\hat{\mathbf{z}} \in \Lambda(\hat{\mathbf{a}}) \setminus \{\mathbf{0}\}$, *and* $(n-1)$*-dimensional hyperplane* $\mathcal{H} \subset \operatorname{span}(\mathbf{B})$,

$$\Pr\{\mathbf{s} \notin \mathcal{H} \mid \mathbf{a} = \hat{\mathbf{a}}, \mathbf{z} = \hat{\mathbf{z}}\} = \Omega(1).$$

Lemma 6.10. *If* $n\|\mathbf{S}\|\lambda_1(\mathcal{L}(\mathbf{L}_n)) \geq 6\alpha(n)\hat{\zeta}(\mathcal{L}(\mathbf{B}))$, (6.4) *holds true, and function* $\alpha(n)$ *satisfies* $\alpha(n) = \omega(n\sqrt{k(n)}\beta(n)\rho(\mathcal{L}(\mathbf{L}_n)))$, *then for any* $\hat{\mathbf{a}} \in G_n^{m(n)}$ *and* $\hat{\mathbf{z}} \in \Lambda(\hat{\mathbf{a}})$,

$$\operatorname{Exp}[\|\mathbf{s}\| \mid \mathbf{a} = \hat{\mathbf{a}}, \mathbf{z} = \hat{\mathbf{z}}] = o\left(\frac{\|\hat{\mathbf{z}}\| \cdot \|\mathbf{S}\|}{\beta(n)}\right) \cdot \sqrt{1 + \frac{m(n)k(n)}{2^{k(n)}}}.$$

*In particular, for any polynomially bounded $m(n) = n^{O(1)}$ and superlogarithmic function $k(n) = \omega(\log n)$,*

$$\mathrm{Exp}[\|\mathbf{s}\| \mid \mathbf{a} = \hat{\mathbf{a}}, \mathbf{z} = \hat{\mathbf{z}}] = o\left(\frac{\|\hat{\mathbf{z}}\| \cdot \|\mathbf{S}\|}{\beta(n)}\right).$$

We complete the proof of the theorem by showing that if $k(n)$ is appropriately chosen, then the hypotheses of Lemmas 6.8, 6.9, and 6.10 are satisfied. Notice that from the definition of $\alpha(n) = n\lambda_1(\mathcal{L}(\mathbf{L}_n))\gamma(n)/12$ and the assumption that $\|\mathbf{S}\| > \gamma(n)\hat{\zeta}(\mathcal{L}(\mathbf{B}))$, we immediately get

$$12\alpha(n)\hat{\zeta}(\mathcal{L}(\mathbf{B})) = n\lambda_1(\mathcal{L}(\mathbf{L}_n))\gamma(n)\hat{\zeta}(\mathcal{L}(\mathbf{B})) < n\lambda_1(\mathcal{L}(\mathbf{L}_n))\|\mathbf{S}\|. \tag{6.6}$$

So, the first condition in Lemmas 6.8, 6.9, and 6.10 is satisfied. We already observed that (6.4) follows from (6.2) and (6.3). In order to satisfy the third hypothesis of Lemma 6.10, we set

$$k(n) = \frac{\gamma(n) \cdot \sqrt{\log n}}{\beta(n) \cdot \tau(n)} = \omega(\log n). \tag{6.7}$$

Solving (6.7) for $\gamma(n) = k(n)\beta(n)\tau(n)/\sqrt{\log n}$ and substituting this value in the definition of $\alpha(n)$, we get

$$\begin{aligned}\alpha(n) &= \frac{n\lambda_1(\mathcal{L}(\mathbf{L}_n))}{12}\frac{k(n)\beta(n)\tau(n)}{\sqrt{\log n}} \\ &\geq \omega(n\sqrt{k(n)}\beta(n)\rho(\mathcal{L}(\mathbf{L}_n))),\end{aligned}$$

where we have used the perfectness condition $\rho(\mathcal{L}(\mathbf{L}_n)) \leq \tau(n)\lambda_1(\mathcal{L}(\mathbf{L}_n))/2$ and the fact that $k(n)/\sqrt{\log n} = \sqrt{k(n)/\log n}\sqrt{k(n)} = \omega(\sqrt{k(n)})$. This proves that for any polynomially bounded function $m(n) = n^{O(1)}$, and $k(n)$ as defined in (6.7), the hypotheses of Lemmas 6.8, 6.9, and Lemma 6.10 are satisfied, and algorithm $\mathcal{A}$ satisfies the conditions in Lemma 6.4. Therefore, for any (possibly probabilistic) oracle $\mathcal{F}$ solving $\mathrm{HSIS}_{G,m,\beta}$ on the average with nonnegligible probability, $\mathcal{A}^{\mathcal{F}}$ solves $\mathrm{IncGIVP}_{\gamma}^{\hat{\zeta}}$ in the worst case with high probability. $\square$

**6.3. The sampling procedure.** In this subsection we give a simple sampling procedure that satisfies the conditions in Lemma 6.6. Then, we establish some additional properties of the output of the sampling procedure that will be useful in section 6.4. The sampling procedure is illustrated in Figure 2.

*Proof of Lemma* 6.6. We first show how to achieve the first two properties in the lemma. Choose integers

$$d_1, \ldots, d_n \in \{1, \ldots, \det(\mathcal{L}(\mathbf{C}))/\det(\mathcal{L}(\mathbf{B}))\}$$

uniformly at random and let $\mathbf{v}'' = \sum_i d_i\mathbf{b}_i \in \mathcal{L}(\mathbf{B})$. By Proposition 2.9, $[\mathbf{v}'']_{\mathbf{C}}$ is distributed uniformly at random, in $\mathcal{L}(\mathbf{B})/\mathcal{L}(\mathbf{C})$. Then, compute $\mathbf{w}'' = \mathrm{CVP}_{\mathbf{L}}(\psi^{-1}(\mathbf{v}''))$. Clearly, $\psi^{-1}(\mathbf{v}'')$ belongs to the Voronoi cell $\bar{\mathcal{V}}(\mathbf{w}'', \mathcal{L}(\mathbf{L}_n))$. So, the pair $(\mathbf{v}'', \mathbf{w}'')$ satisfies the first two properties.

Now, choose $b \in \{0, 1\}$ uniformly at random and set $\mathbf{v}' = (-1)^b\mathbf{v}''$ and $\mathbf{w}' = (-1)^b\mathbf{w}''$. Clearly, for any $\mathbf{v}''$ and $\mathbf{w}''$, the distribution of $\mathbf{v}' - \psi(\mathbf{w}') = (-1)^b(\mathbf{v}'' - \psi(\mathbf{w}''))$ is symmetric about the origin. So, $(\mathbf{v}', \mathbf{w}')$ satisfies the third property. We need to check that the first two properties are preserved. Since $[\mathbf{v}'']_{\mathbf{C}}$ is uniformly distributed, $[-\mathbf{v}'']_{\mathbf{C}} = -[\mathbf{v}'']_{\mathbf{C}}$ is also uniform. It follows that $[\mathbf{v}']_{\mathbf{C}}$ is uniformly distributed because $\mathbf{v}'$ is a convex combination of distributions $\mathbf{v}''$ and $-\mathbf{v}''$. Finally, since closed Voronoi cells of a lattice are symmetric,

$$\mathbf{v}' - \psi(\mathbf{w}') = (-1)^b(\mathbf{v}'' - \psi(\mathbf{w}'')) \in (-1)^b\bar{\mathcal{V}}(\mathcal{L}(\mathbf{L}_n)) = \bar{\mathcal{V}}(\mathcal{L}(\mathbf{L}_n)).$$

This proves that $(\mathbf{v}', \mathbf{w}')$ satisfies the first three properties in the lemma.

In order to also achieve the fourth property, set $\mathbf{w} = (\mathbf{w}' \bmod \mathbf{M}_n)$ and $\mathbf{v} = (\mathbf{v}' - \psi(\mathbf{w}' - \mathbf{w}))$. Property $\mathbf{w} \in \mathcal{P}(\mathbf{M}_n)$ immediately follows by the definition of $\mathbf{w}$. We show that the first three properties are preserved. By the definition of $\mathbf{v}$, we have $\mathbf{v}' - \mathbf{v} = \psi(\mathbf{w}' - \mathbf{w})$ and $\mathbf{v} - \psi(\mathbf{w}) = \mathbf{v}' - \psi(\mathbf{w}')$. So, the second and third properties are satisfied because they depend only on $\mathbf{v} - \psi(\mathbf{w})$. In order to prove the first property, notice that $\mathbf{w}' - \mathbf{w} = \mathbf{w}' - (\mathbf{w}' \bmod \mathbf{M}_n) \in \mathcal{L}(\mathbf{M}_n)$. Therefore, $\mathbf{v}' - \mathbf{v} \in \psi(\mathcal{L}(\mathbf{M}_n)) = \mathcal{L}(\mathbf{C})$, and $[\mathbf{v}]_{\mathbf{C}} = [\mathbf{v}']_{\mathbf{C}}$, proving that $[\mathbf{v}]_{\mathbf{C}}$ is distributed identically to $[\mathbf{v}']_{\mathbf{C}}$. ∎

The sampling procedure produces vectors $\mathbf{v} \in \mathcal{L}(\mathbf{B})$ such that $[\mathbf{v}]_{\mathbf{C}}$ is distributed uniformly at random over the group $\mathcal{L}(\mathbf{B})/\mathcal{L}(\mathbf{C})$. However, vector $\mathbf{v}$ (before the reduction modulo $\mathbf{C}$) is not necessarily uniformly distributed over any set of lattice vectors. (This is due to lattice points $\mathbf{v} \in \mathcal{L}(\mathbf{B})$ such that $\psi^{-1}(\mathbf{v})$ lies on the boundary of Voronoi cells $\mathcal{V}(\mathbf{w}, \mathcal{L}(\mathbf{L}_n))$.) In the next lemma, we give simple upper and lower bounds on the probability of outputting any specific vector $\mathbf{v} \in \mathcal{L}(\mathbf{B})$.

Lemma 6.11. *Let* $(\mathbf{w}, \mathbf{v})$ *be generated according to a sampling procedure of Lemma* 6.6. *Then, for any* $\hat{\mathbf{v}} \in \mathcal{L}(\mathbf{B})$, $\Pr\{\mathbf{v} = \hat{\mathbf{v}}\} \le \det(\mathcal{L}(\mathbf{B}))/\det(\mathcal{L}(\mathbf{C}))$. *Moreover, if* $\psi^{-1}(\hat{\mathbf{v}})$ *belongs to the interior of a Voronoi cell* $\mathcal{V}^o(\hat{\mathbf{w}}, \mathcal{L}(\mathbf{L}_n))$ *for some* $\hat{\mathbf{w}} \in \mathcal{L}(\mathbf{L}_n) \cap \mathcal{P}(\mathbf{M}_n)$, *then* $\Pr\{\mathbf{v} = \hat{\mathbf{v}}\} = \det(\mathcal{L}(\mathbf{B}))/\det(\mathcal{L}(\mathbf{C}))$.

*Proof.* The upper bound is easy: for any $\hat{\mathbf{v}} \in \mathcal{L}(\mathbf{B})$,

$$\Pr\{\mathbf{v} = \hat{\mathbf{v}}\} \le \Pr\{[\mathbf{v}]_{\mathbf{C}} = [\hat{\mathbf{v}}]_{\mathbf{C}}\} = \det(\mathcal{L}(\mathbf{B}))/\det(\mathcal{L}(\mathbf{C}))$$

because $[\mathbf{v}]_{\mathbf{C}}$ is uniformly distributed over a quotient group $\mathcal{L}(\mathbf{B})/\mathcal{L}(\mathbf{C})$ whose size equals $\det(\mathcal{L}(\mathbf{C}))/\det(\mathcal{L}(\mathbf{B}))$.

Now assume $\psi^{-1}(\hat{\mathbf{v}}) \in \mathcal{V}^o(\hat{\mathbf{w}}, \mathcal{L}(\mathbf{L}_n))$ for some $\hat{\mathbf{w}} \in \mathcal{L}(\mathbf{L}_n) \cap \mathcal{P}(\mathbf{M}_n)$. We claim that if $[\mathbf{v}]_{\mathbf{C}} = [\hat{\mathbf{v}}]_{\mathbf{C}}$, then $\mathbf{v} = \hat{\mathbf{v}}$, and therefore

$$\Pr\{\mathbf{v} = \hat{\mathbf{v}}\} \ge \Pr\{[\mathbf{v}]_{\mathbf{C}} = [\hat{\mathbf{v}}]_{\mathbf{C}}\} = \det(\mathcal{L}(\mathbf{B}))/\det(\mathcal{L}(\mathbf{C})).$$

Let $[\mathbf{v}]_{\mathbf{C}} = [\hat{\mathbf{v}}]_{\mathbf{C}}$, i.e., $\mathbf{v} - \hat{\mathbf{v}} \in \mathcal{L}(\mathbf{C})$. It follows that vector

$$\mathbf{y} = \psi^{-1}(\mathbf{v}) - \psi^{-1}(\hat{\mathbf{v}}) = \psi^{-1}(\mathbf{v} - \hat{\mathbf{v}})$$

belongs to lattice $\psi^{-1}(\mathcal{L}(\mathbf{C})) = \mathcal{L}(\mathbf{M}_n)$. Since $\mathcal{L}(\mathbf{M}_n)$ is a sublattice of $\mathcal{L}(\mathbf{L}_n)$, and $\hat{\mathbf{w}} \in \mathcal{L}(\mathbf{L}_n)$, $\hat{\mathbf{w}} + \mathbf{y}$ is also a lattice point in $\mathcal{L}(\mathbf{L}_n)$. Consider the open Voronoi cells $\mathcal{V}^o(\hat{\mathbf{w}}, \mathcal{L}(\mathbf{L}_n))$ and $\mathcal{V}^o(\hat{\mathbf{w}} + \mathbf{y}, \mathcal{L}(\mathbf{L}_n))$. Using the definition of $\mathbf{y}$ and the hypothesis $\psi^{-1}(\hat{\mathbf{v}}) \in \mathcal{V}^o(\hat{\mathbf{w}}, \mathcal{L}(\mathbf{L}_n))$, we get

$$\psi^{-1}(\mathbf{v}) = \psi^{-1}(\hat{\mathbf{v}}) + \mathbf{y} \in \mathcal{V}^o(\hat{\mathbf{w}}, \mathcal{L}(\mathbf{L}_n)) + \mathbf{y} = \mathcal{V}^o(\hat{\mathbf{w}} + \mathbf{y}, \mathcal{L}(\mathbf{L}_n));$$

i.e., $\psi^{-1}(\mathbf{v})$ is closer to $\hat{\mathbf{w}} + \mathbf{y}$ than to any other lattice point in $\mathcal{L}(\mathbf{L}_n)$. But we know from Lemma 6.6 that $\psi^{-1}(\mathbf{v})$ belongs to the Voronoi cell $\bar{\mathcal{V}}(\mathbf{w}, \mathcal{L}(\mathbf{L}_n))$; i.e., $\psi^{-1}(\mathbf{v})$ is at least as close to $\mathbf{w} \in \mathcal{L}(\mathbf{L}_n)$ as to any other lattice point. Therefore, it must be

$\mathbf{w} = \hat{\mathbf{w}} + \mathbf{y}$. We also know that both $\mathbf{w}$ and $\hat{\mathbf{w}}$ belong to $\mathcal{P}(\mathbf{M}_n)$, and $\mathbf{y} \in \mathcal{L}(\mathbf{M}_n)$. So, $\mathbf{w} = \hat{\mathbf{w}} + \mathbf{y}$ is possible only if $\mathbf{y} = \mathbf{0}$, which implies $\mathbf{v} = \hat{\mathbf{v}}$. ◻

Lemma 6.11 can be used to establish two important properties of the sampling algorithm of Lemma 6.6. The distribution $[\mathbf{v}]_{\mathbf{C}}$ produced by the sampling algorithm is uniform. However, $[\mathbf{w}]_{\mathbf{M}_n}$ is not in general uniformly distributed over $G_n$. The first property is that, provided $\|\mathbf{S}\|$ is large enough, the distribution of $[\mathbf{w}]_{\mathbf{M}_n}$ is relatively close to uniform.

LEMMA 6.12. *Let* $(\mathbf{w}, \mathbf{v})$ *be generated according to the sampling procedure of Lemma* 6.6. *If* $n\|\mathbf{S}\|\lambda_1(\mathcal{L}(\mathbf{L}_n)) \geq 6\alpha(n)\hat{\zeta}(\mathcal{L}(\mathbf{B}))$ *and* (6.4) *holds true, then for any group element* $g \in G_n$,

$$\Pr\{[\mathbf{w}]_{\mathbf{M}_n} = g\} \approx \frac{1}{\#G_n}.$$

*Proof.* Fix group element $g$, and let $\hat{\mathbf{w}}$ be the unique lattice point in $\mathcal{L}(\mathbf{L}_n) \cap \mathcal{P}(\mathbf{M}_n)$ such that $[\hat{\mathbf{w}}]_{\mathbf{M}_n} = g$. Since $\mathbf{w} \in \mathcal{P}(\mathbf{M}_n)$, $[\mathbf{w}]_{\mathbf{M}_n} = g$ if and only if $\mathbf{w} = \hat{\mathbf{w}}$. We estimate the probability that $\mathbf{w} = \hat{\mathbf{w}}$.

Notice that if $\mathbf{v} \in \psi(\mathcal{V}^o(\hat{\mathbf{w}}, \mathcal{L}(\mathbf{L}_n)))$, then $\mathbf{w} = \hat{\mathbf{w}}$. Therefore,

$$\Pr\{\mathbf{w} = \hat{\mathbf{w}}\} \geq \sum_{\hat{\mathbf{v}} \in \psi(\mathcal{V}^o(\hat{\mathbf{w}}, \mathcal{L}(\mathbf{L}_n))) \cap \mathcal{L}(\mathbf{B})} \Pr\{\mathbf{v} = \hat{\mathbf{v}}\}.$$

By Lemma 6.11, for any $\hat{\mathbf{v}} \in \psi(\mathcal{V}^o(\hat{\mathbf{w}}, \mathcal{L}(\mathbf{L}_n))) \cap \mathcal{L}(\mathbf{B})$,

$$\Pr\{\mathbf{v} = \hat{\mathbf{v}}\} = \det(\mathcal{L}(\mathbf{B}))/\det(\mathcal{L}(\mathbf{C})).$$

So,

$$\Pr\{\mathbf{w} = \hat{\mathbf{w}}\} \geq \frac{\det(\mathcal{L}(\mathbf{B}))}{\det(\mathcal{L}(\mathbf{C}))} \cdot \#(\psi(\mathcal{V}^o(\hat{\mathbf{w}}, \mathcal{L}(\mathbf{L}_n))) \cap \mathcal{L}(\mathbf{B})).$$

Similarly, if $\mathbf{w} = \hat{\mathbf{w}}$, then $\mathbf{v} \in \psi(\bar{\mathcal{V}}(\hat{\mathbf{w}}, \mathcal{L}(\mathbf{L}_n)))$. Therefore,

$$\begin{aligned}\Pr\{\mathbf{w} = \hat{\mathbf{w}}\} &\leq \sum_{\hat{\mathbf{v}} \in \psi(\bar{\mathcal{V}}(\hat{\mathbf{w}}, \mathcal{L}(\mathbf{L}_n))) \cap \mathcal{L}(\mathbf{B})} \Pr\{\mathbf{v} = \hat{\mathbf{v}}\} \\ &\leq \frac{\det(\mathcal{L}(\mathbf{B}))}{\det(\mathcal{L}(\mathbf{C}))} \cdot \#(\psi(\bar{\mathcal{V}}(\hat{\mathbf{w}}, \mathcal{L}(\mathbf{L}_n))) \cap \mathcal{L}(\mathbf{B})).\end{aligned}$$

In order to complete the proof, we need to estimate the number of lattice points from $\mathcal{L}(\mathbf{B})$ that belong to $\psi(\mathcal{V}^o(\hat{\mathbf{w}}, \mathcal{L}(\mathbf{L}_n)))$ and $\psi(\bar{\mathcal{V}}(\hat{\mathbf{w}}, \mathcal{L}(\mathbf{L}_n)))$. Since $\mathcal{V}^o(\hat{\mathbf{w}}, \mathcal{L}(\mathbf{L}_n))$ contains an open sphere of radius $\lambda_1(\mathcal{L}(\mathbf{L}_n))/2$, using (6.4) we get that the set $\psi(\mathcal{V}^o(\hat{\mathbf{w}}, \mathcal{L}(\mathbf{L}_n)))$ (and therefore, also $\psi(\bar{\mathcal{V}}(\hat{\mathbf{w}}, \mathcal{L}(\mathbf{L}_n))))$ contains a sphere of radius

$$\frac{n\|\mathbf{S}\|}{3\alpha(n)} \cdot \frac{\lambda_1(\mathcal{L}(\mathbf{L}_n))}{2} \geq \hat{\zeta}(\mathcal{L}(\mathbf{B})).$$

Therefore, by definition of the generalized uniform radius $\hat{\zeta}(\mathcal{L}(\mathbf{B}))$, the number of lattice points in $\psi(\mathcal{V}^o(\hat{\mathbf{w}}, \mathcal{L}(\mathbf{L}_n)))$ (and $\psi(\bar{\mathcal{V}}(\hat{\mathbf{w}}, \mathcal{L}(\mathbf{L}_n))))$ is approximately equal to

$$\frac{\operatorname{vol}(\psi(\mathcal{V}^o(\hat{\mathbf{w}}, \mathcal{L}(\mathbf{L}_n))))}{\det(\mathcal{L}(\mathbf{B}))} = \frac{\operatorname{vol}(\psi(\bar{\mathcal{V}}(\hat{\mathbf{w}}, \mathcal{L}(\mathbf{L}_n))))}{\det(\mathcal{L}(\mathbf{B}))} = \frac{\operatorname{vol}(\psi(\mathcal{V}(\hat{\mathbf{w}}, \mathcal{L}(\mathbf{L}_n))))}{\det(\mathcal{L}(\mathbf{B}))}.$$

Combining this estimate with the upper and lower bounds on the probability that $\mathbf{w} = \hat{\mathbf{w}}$, we get

$$\begin{aligned}\Pr\{[\mathbf{w}]_{\mathbf{M}_n} = g\} &\approx \frac{\det(\mathcal{L}(\mathbf{B}))}{\det(\mathcal{L}(\mathbf{C}))} \cdot \frac{\mathrm{vol}(\psi(\mathcal{V}(\hat{\mathbf{w}}, \mathcal{L}(\mathbf{L}_n))))}{\det(\mathcal{L}(\mathbf{B}))} \\ &= \frac{\mathrm{vol}(\psi(\mathcal{V}(\hat{\mathbf{w}}, \mathcal{L}(\mathbf{L}_n))))}{\det(\psi(\mathcal{L}(\mathbf{M}_n)))} \\ &= \frac{\mathrm{vol}(\mathcal{V}(\hat{\mathbf{w}}, \mathcal{L}(\mathbf{L}_n)))}{\det(\mathcal{L}(\mathbf{M}_n))} \\ &= \frac{\det(\mathcal{L}(\mathbf{L}_n))}{\det(\mathcal{L}(\mathbf{M}_n))} = \frac{1}{\#G_n}. \qquad \square\end{aligned}$$

The second property implies that, provided $\|\mathbf{S}\|$ is large enough, the distribution of $\mathbf{v} - \psi(\mathbf{w})$, as generated by the sampling procedure, is not concentrated over any fixed $(n-1)$-dimensional hyperplane. In fact, we prove a stronger property and show that $\mathbf{v} - \psi(\mathbf{w})$ belongs to any of the two half-spaces defined by the hyperplane with high probability. This is true even for the conditional distribution of $\mathbf{v} - \psi(\mathbf{w})$ given $\mathbf{w}$.

LEMMA 6.13. *Let* $(\mathbf{w}, \mathbf{v})$ *be generated according to the sampling procedure of Lemma* 6.6. *If* $n\|\mathbf{S}\|\lambda_1(\mathcal{L}(\mathbf{L}_n)) \geq 12\alpha(n)\hat{\zeta}(\mathcal{L}(\mathbf{B}))$ *and* (6.4) *holds true, then for any* $\mathbf{h} \in \mathrm{span}(\mathbf{B}) \setminus \{\mathbf{0}\}$ *and* $g \in G_n$,

$$\Pr\{\mathbf{h}^T \cdot (\mathbf{v} - \psi(\mathbf{w})) > 0 \mid [\mathbf{w}]_{\mathbf{M}_n} = g\} \geq \frac{1}{6}.$$

*Proof.* Fix group element $g$, and let $\hat{\mathbf{w}}$ be the unique lattice point in $\mathcal{L}(\mathbf{L}_n) \cap \mathcal{P}(\mathbf{M}_n)$ such that $[\hat{\mathbf{w}}]_{\mathbf{M}_n} = g$. Since Lemma 6.6 guarantees $\mathbf{w} \in \mathcal{P}(\mathbf{M}_n)$, condition $[\mathbf{w}]_{\mathbf{M}_n} = g$ is equivalent to $\mathbf{w} = \hat{\mathbf{w}}$. Let $\mathcal{Q} = \{\mathbf{x} \in \mathcal{V}^o(\hat{\mathbf{w}}, \mathcal{L}(\mathbf{L}_n)) \colon \mathbf{h}^T \cdot \psi(\mathbf{x} - \hat{\mathbf{w}}) > 0\}$ be one of the two (open) halves of the Voronoi cell $\mathcal{V}^o(\hat{\mathbf{w}}, \mathcal{L}(\mathbf{L}_n))$ defined by the hyperplane $\mathbf{h}^T \cdot \psi(\mathbf{x}) = \mathbf{h}^T \cdot \psi(\hat{\mathbf{w}})$. (See Figure 3.) First we estimate the probability

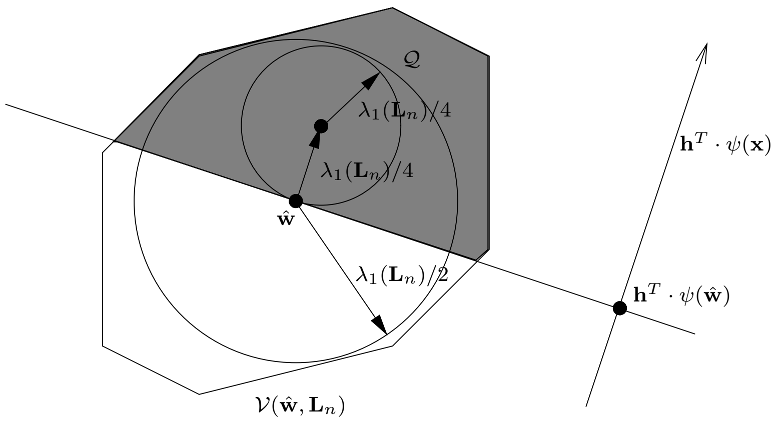

$\mathcal{Q}$
$\lambda_1(\mathbf{L}_n)/4$
$\lambda_1(\mathbf{L}_n)/4$
$\hat{\mathbf{w}}$
$\lambda_1(\mathbf{L}_n)/2$
$\mathbf{h}^T \cdot \psi(\mathbf{x})$
$\mathbf{h}^T \cdot \psi(\hat{\mathbf{w}})$
$\mathcal{V}(\hat{\mathbf{w}}, \mathbf{L}_n)$

FIG. 3. *The conditional distribution of sampled lattice points.*

that $\mathbf{v} \in \psi(\mathcal{Q})$. Since $\mathcal{Q}$ is contained in the open Voronoi cell $\mathcal{V}^o(\hat{\mathbf{w}}, \mathcal{L}(\mathbf{L}_n))$, by Lemma 6.11,

$$\Pr\{\mathbf{v} \in \psi(\mathcal{Q})\} = \sum_{\hat{\mathbf{v}} \in \psi(\mathcal{Q}) \cap \mathcal{L}(\mathbf{B})} \Pr\{\mathbf{v} = \hat{\mathbf{v}}\} = \frac{\det(\mathcal{L}(\mathbf{B}))}{\det(\mathcal{L}(\mathbf{C}))} \cdot \#(\psi(\mathcal{Q}) \cap \mathcal{L}(\mathbf{B})).$$

Notice that $\mathcal{Q}$ contains an open sphere of radius $\lambda_1(\mathcal{L}(\mathbf{L}_n))/4$. (See Figure 3.) Therefore, by (6.4), $\psi(\mathcal{Q})$ contains a sphere of radius

$$\frac{n\|\mathbf{S}\|}{3\alpha(n)} \cdot \frac{\lambda_1(\mathcal{L}(\mathbf{L}_n))}{4} \geq \hat{\zeta}(\mathcal{L}(\mathbf{B})).$$

By definition of $\hat{\zeta}(\mathcal{L}(\mathbf{B}))$, the number of lattice points in $\psi(\mathcal{Q})$ satisfies

$$\#(\psi(\mathcal{Q}) \cap \mathcal{L}(\mathbf{B})) \approx \frac{\operatorname{vol}(\psi(\mathcal{Q}))}{\det(\mathcal{L}(\mathbf{B}))} = \frac{1}{2} \frac{\operatorname{vol}(\psi(\mathcal{V}(\hat{\mathbf{w}}, \mathcal{L}(\mathbf{L}_n))))}{\det(\mathcal{L}(\mathbf{B}))}$$

and

$$\begin{aligned}\Pr\{\mathbf{v} \in \psi(\mathcal{Q})\} &\approx \frac{1}{2} \frac{\det(\mathcal{L}(\mathbf{B}))}{\det(\mathcal{L}(\mathbf{C}))} \cdot \frac{\operatorname{vol}(\psi(\mathcal{V}(\hat{\mathbf{w}}, \mathcal{L}(\mathbf{L}_n))))}{\det(\mathcal{L}(\mathbf{B}))} \\ &= \frac{1}{2} \frac{\operatorname{vol}(\psi(\mathcal{V}(\hat{\mathbf{w}}, \mathcal{L}(\mathbf{L}_n))))}{\det(\psi(\mathcal{L}(\mathbf{M}_n)))} \\ &= \frac{1}{2} \frac{\operatorname{vol}(\mathcal{V}(\hat{\mathbf{w}}, \mathcal{L}(\mathbf{L}_n)))}{\det(\mathcal{L}(\mathbf{M}_n))} \\ &= \frac{1}{2 \cdot \# G_n}.\end{aligned}$$

Notice that if $\mathbf{v} \in \psi(\mathcal{Q})$, then $\mathbf{w} = \hat{\mathbf{w}}$ and $\mathbf{h}^T \cdot (\mathbf{v} - \psi(\hat{\mathbf{w}})) > 0$. Therefore,

$$\Pr\{[\mathbf{w}]_{\mathbf{M}_n} = g \wedge \mathbf{h}^T \cdot (\mathbf{v} - \psi(\hat{\mathbf{w}})) > 0\} \geq \Pr\{\mathbf{v} \in \psi(\mathcal{Q})\} \geq \frac{1}{4 \cdot \# G_n}.$$

We can now compute the conditional probability,

$$\begin{aligned}\Pr\{\mathbf{h}^T \cdot (\mathbf{v} - \psi(\mathbf{w})) > 0 \mid [\mathbf{w}]_{\mathbf{M}_n} = g\} &= \frac{\Pr\{\mathbf{h}^T \cdot (\mathbf{v} - \psi(\mathbf{w})) > 0 \wedge [\mathbf{w}]_{\mathbf{M}_n} = g\}}{\Pr\{[\mathbf{w}]_{\mathbf{M}_n} = g\}} \\ &\geq \frac{1}{4 \cdot \# G_n \cdot \Pr\{[\mathbf{w}]_{\mathbf{M}_n} = g\}}.\end{aligned}$$

Using Lemma 6.12, $\Pr\{[\mathbf{w}]_{\mathbf{M}_n} = g\} \lesssim 1/\# G_n$, i.e., $\# G_n \cdot \Pr\{[\mathbf{w}]_{\mathbf{M}_n} = g\} \leq 3/2$. Substituting in the previous inequality we get

$$\Pr\{\mathbf{h}^T \cdot (\mathbf{v} - \psi(\mathbf{w})) > 0 \mid [\mathbf{w}]_{\mathbf{M}_n} = g\} \geq \frac{1}{6}. \qquad \square$$

**6.4. Proofs of the lemmas.** In this subsection we prove Lemmas 6.8, 6.9, and 6.10 used in the analysis the algorithm $\mathcal{A}^{\mathcal{F}}(\mathbf{B}, \mathbf{S})$ in section 6.2. The intuition behind Lemma 6.8 is that since group elements $a_{i,j}$ are independent and not too far from uniformly distributed, their sums $a_i = \sum_j a_{i,j}$ are extremely close to uniform.

*Proof of Lemma* 6.8. First consider the distribution of a single group element $a_{i,j} = [\mathbf{w}_{i,j}]_{\mathbf{M}_n}$ as output by the sampling procedure. Since $n\|\mathbf{S}\|\lambda_1(\mathcal{L}(\mathbf{L}_n)) \geq$

$6\alpha(n)\hat{\zeta}(\mathcal{L}(\mathbf{B}))$ and (6.4) holds true by assumption, Lemma 6.12 tells us that for any group element $g \in G_n$,

$$\Pr\{a_{i,j} = g\} \approx \frac{1}{\#G_n}.$$

So, the probability distribution of each $a_{i,j}$ is not too far from uniform. Adding up a relatively small number of $a_{i,j}$ we get a group element $a_i = \sum_{j=1}^{k(n)} a_{i,j}$ which is almost uniformly distributed. In particular, by Proposition 2.17, the statistical distance between $a_i$ and a uniformly distributed $u_i \in G$ is at most

$$\Delta(a_i, u_i) \le \frac{1}{2^{k(n)+1}}. \tag{6.8}$$

Since the random variables $a_i$ are independent, by Proposition 2.14 the statistical distance between vector $\mathbf{a} = [a_1, \ldots, a_{m(n)}]^T$ and a uniformly distributed $\mathbf{u} \in G_n^{m(n)}$ is at most

$$\Delta(\mathbf{a}, \mathbf{u}) \le \sum_{i=1}^{m(n)} \Delta(a_i, u_i) \le \frac{m(n)}{2^{k(n)+1}} = \frac{n^{O(1)}}{n^{\omega(1)}} = n^{-\omega(1)}. \qquad \square \tag{6.9}$$

The intuition behind Lemma 6.9 is that since each $\mathbf{y}_{i,j} = \mathbf{v}_{i,j} - \psi(\mathbf{w}_{i,j})$ has the property described in Lemma 6.13, then also $s$ (which is a nonzero linear combination of the $\mathbf{y}_{i,j}$'s) has a similar property.

*Proof of Lemma* 6.9. Fix $\hat{\mathbf{a}} \in G_n^{m(n)}$, $\hat{\mathbf{z}} \in \Lambda(\hat{\mathbf{a}}) \setminus \{\mathbf{0}\}$, and $(n-1)$-dimensional hyperplane $\mathcal{H} \subset \operatorname{span}(\mathbf{B})$. Since $\hat{\mathbf{z}} \neq \mathbf{0}$, there exists a coordinate $i$ such that $\hat{z}_i \neq 0$. Assume without loss of generality that $\hat{z}_1 \neq 0$.

The output of $\mathcal{A}^{\mathcal{F}}(\mathbf{B}, \mathbf{S})$ is a random variable that depends on the randomness of oracle $\mathcal{F}$ and the randomness used during the execution of the sampling procedure in the computation of $(\mathbf{w}_{i,j}, \mathbf{v}_{i,j})$ for $i = 1, \ldots, m(n)$ and $j = 1, \ldots, k(n)$. Fix the randomness of $\mathcal{F}$ and sampling procedure, except for $(i,j) = (1,1)$. Finally, for this remaining run of the sampling procedure, fix the value of $\mathbf{w}_{1,1}$ and consider the conditional distribution of $\mathbf{v}_{1,1}$. Notice that this uniquely determines

(i) the values of $\mathbf{v}_{i,j}$ for all $(i,j) \neq (1,1)$,
(ii) the values of $\mathbf{w}_{i,j}$ for all $i, j$,
(iii) the value of $\mathbf{a} = \sum_{i,j} [\mathbf{w}_{i,j}]_{\mathbf{M}_n} \mathbf{e}_i$, and
(iv) the value of $\mathbf{z} = \mathcal{F}(\mathbf{a})$.

We prove a stronger statement than the one in the lemma. Namely, we show that the conditional probability

$$\Pr\{\mathbf{s} \notin \mathcal{H} \mid \mathbf{a} = \hat{\mathbf{a}}, \mathbf{z} = \hat{\mathbf{z}}, \forall(i,j).\mathbf{w}_{i,j} = \hat{\mathbf{w}}_{i,j}, \forall(i,j) \neq (1,1).\mathbf{v}_{i,j} = \hat{\mathbf{v}}_{i,j}\}$$

is at least $1/6$. Notice that this probability depends only on the conditional distribution of $\mathbf{v}_{1,1}$, because all other vectors are fixed by conditioning. Averaging over all cases such that $\mathbf{a} = \hat{\mathbf{a}}$ and $\mathbf{z} = \hat{\mathbf{z}}$, we get that $\Pr\{\mathbf{s} \notin \mathcal{H} \mid \mathbf{a} = \hat{\mathbf{a}}, \mathbf{z} = \hat{\mathbf{z}}\} \ge 1/6$.

For any fixed values $\hat{\mathbf{w}}_{i,j}$, $\hat{\mathbf{v}}_{i,j}$, and $\hat{\mathbf{a}}$, define the vector

$$\mathbf{y} = \sum_{(i,j) \neq (1,1)} \hat{z}_i \cdot (\hat{\mathbf{v}}_{i,j} - \psi(\hat{\mathbf{w}}_{i,j})).$$

Notice that, given $\mathbf{z} = \hat{\mathbf{z}}$, $\mathbf{w}_{i,j} = \hat{\mathbf{w}}_{i,j}$ for all $i,j$, and $\mathbf{v}_{i,j} = \hat{\mathbf{v}}_{i,j}$ for all $(i,j) \neq (1,1)$,

$$\mathbf{s} = \sum_{i,j} z_i \cdot (\mathbf{v}_{i,j} - \psi(\mathbf{w}_{i,j})) = \mathbf{y} + \hat{z}_1 \cdot (\mathbf{v}_{1,1} - \psi(\hat{\mathbf{w}}_{1,1})).$$

We want to bound the conditional probability that $\mathbf{s} \in \mathcal{H}$ or, equivalently,

$$\mathbf{v}_{1,1} - \psi(\hat{\mathbf{w}}_{1,1}) \in \frac{\mathcal{H} - \mathbf{y}}{\hat{z}_1} = \mathcal{H}'.$$

Let $\mathbf{h} \in \operatorname{span}(\mathbf{B})$ be a vector orthogonal to $\mathcal{H}'$ such that $\mathbf{h}^T \cdot \mathbf{x} \leq 0$ for any $\mathbf{x} \in \mathcal{H}'$. (Notice that since $\mathbf{h}$ is orthogonal to $\mathcal{H}'$, the function $\mathbf{x} \mapsto \mathbf{h}^T \cdot \mathbf{x}$ is constant over $\mathcal{H}'$.) Since $\mathbf{h}^T \cdot \mathbf{x} > 0$ implies $\mathbf{x} \notin \mathcal{H}'$ for all vectors $\mathbf{x}$, the conditional probability that $\mathbf{v}_{1,1} - \psi(\hat{\mathbf{w}}_{1,1}) \notin \mathcal{H}'$ is at least as big as the conditional probability that $\mathbf{h}^T \cdot (\mathbf{v}_{1,1} - \psi(\hat{\mathbf{w}}_{1,1})) > 0$. Since $n\|\mathbf{S}\|\lambda_1(\mathcal{L}(\mathbf{L}_n)) \geq 12\alpha(n)\hat{\zeta}(\mathcal{L}(\mathbf{B}))$ and (6.4) holds true by assumption, Lemma 6.13 tells us that the latter probability is at least 1/6. □

The intuition behind Lemma 6.10 is that vector $\mathbf{s}$ is short because it is a linear combination (with small coefficients $z_i$) of short vectors $\mathbf{y}_{i,j}$ lying within $\psi(\bar{\mathcal{V}}(\mathcal{L}(\mathbf{L}_n)))$. A bound on the length of $\|\mathbf{s}\|$ can be easily computed using the triangle inequality. This would lead to a result similar to the one in Theorem 6.5, but with a larger (by approximately $\sqrt{m(n) \cdot k(n)}$) value of $\gamma(n)$. Here we use cancellations between the $\mathbf{y}_{i,j}$ vectors to prove a better bound.

*Proof of Lemma* 6.10. First, we prove an upper bound on the length of $\mathbf{y}_{i,j} = \mathbf{v}_{i,j} - \psi(\mathbf{w}_{i,j})$ for all $(\mathbf{w}_{i,j}, \mathbf{v}_{i,j})$ in the range of the sampling algorithm of Lemma 6.6. We know from Lemma 6.6 that $\psi^{-1}(\mathbf{v}_{i,j}) \in \bar{\mathcal{V}}(\mathbf{w}_{i,j}, \mathcal{L}(\mathbf{L}_n))$, and therefore $\|\psi^{-1}(\mathbf{v}_{i,j}) - \mathbf{w}_{i,j}\| \leq \rho(\mathcal{L}(\mathbf{L}_n))$. Using (6.4) we immediately get that $\mathbf{y}_{i,j} = \psi(\psi^{-1}(\mathbf{v}_{i,j}) - \mathbf{w}_{i,j})$ has length at most

$$\|\mathbf{y}_{i,j}\| \leq \frac{3n\|\mathbf{S}\| \cdot \rho(\mathcal{L}(\mathbf{L}_n))}{\alpha(n)}. \tag{6.10}$$

Substituting $\alpha(n) = \omega(n\sqrt{k(n)}\beta(n)\rho(\mathcal{L}(\mathbf{L}_n)))$ into (6.10), we get

$$\|\mathbf{y}_{i,j}\| \leq \frac{3\|\mathbf{S}\|}{\omega(\sqrt{k(n)}\beta(n))} = o\left(\frac{\|\mathbf{S}\|}{\sqrt{k(n)}\beta(n)}\right). \tag{6.11}$$

By the triangle inequality and Cauchy–Schwarz, it immediately follows that

$$\|\mathbf{s}\| \leq \sum_{i=1}^{m(n)} |z_i| \sum_{j=1}^{k(n)} \|\mathbf{y}_{i,j}\| \leq \sqrt{m(n)k(n)} \cdot o\left(\frac{\|\mathbf{S}\| \cdot \|\mathbf{z}\|}{\beta(n)}\right).$$

We want to prove a better (probabilistic) bound by computing the conditional expectation of $\|\mathbf{s}\|^2$. Using the definition of $\mathbf{s}$ (6.5), we get

$$\begin{aligned}
\operatorname{Exp}[\|\mathbf{s}\|^2 \mid \mathbf{a} = \hat{\mathbf{a}}, \mathbf{z} = \hat{\mathbf{z}}] &= \operatorname{Exp}\left[\left\langle \sum_{i=1}^{m(n)} \hat{z}_i \sum_{j=1}^{k(n)} \mathbf{y}_{i,j}, \sum_{i'=1}^{m(n)} \hat{z}_{i'} \sum_{j'=1}^{k(n)} \mathbf{y}_{i',j'} \right\rangle \middle| \mathbf{a} = \hat{\mathbf{a}}, \mathbf{z} = \hat{\mathbf{z}}\right] \\
&= \sum_{i,i'} \hat{z}_i \hat{z}_{i'} \sum_{j,j'} \operatorname{Exp}[\langle \mathbf{y}_{i,j}, \mathbf{y}_{i',j'} \rangle \mid \mathbf{a} = \hat{\mathbf{a}}, \mathbf{z} = \hat{\mathbf{z}}] \\
&\leq \sum_{i,i'} |\hat{z}_i \hat{z}_{i'}| \sum_{j,j'} |\operatorname{Exp}[\langle \mathbf{y}_{i,j}, \mathbf{y}_{i',j'} \rangle \mid \mathbf{a} = \hat{\mathbf{a}}, \mathbf{z} = \hat{\mathbf{z}}]|.
\end{aligned}$$

We bound each term $\operatorname{Exp}[\langle \mathbf{y}_{i,j}, \mathbf{y}_{i',j'} \rangle \mid \mathbf{a} = \hat{\mathbf{a}}, \mathbf{z} = \hat{\mathbf{z}}]$ separately. Notice that, given $\mathbf{a} = \hat{\mathbf{a}}$, vector $\mathbf{z} = \mathcal{F}(\mathbf{a}) = \mathcal{F}(\hat{\mathbf{a}})$ depends only on the randomness of oracle $\mathcal{F}$. Therefore, given $\mathbf{a} = \hat{\mathbf{a}}$, $\mathbf{z}$ is independent from $\mathbf{y}_{i,j}$ and $\mathbf{y}_{i',j'}$, and

$$\operatorname{Exp}[\langle \mathbf{y}_{i,j}, \mathbf{y}_{i',j'} \rangle \mid \mathbf{a} = \hat{\mathbf{a}}, \mathbf{z} = \hat{\mathbf{z}}] = \operatorname{Exp}[\langle \mathbf{y}_{i,j}, \mathbf{y}_{i',j'} \rangle \mid \mathbf{a} = \hat{\mathbf{a}}]. \tag{6.12}$$

We will bound the value of (6.12) and show that

$$\text{Exp}[\langle \mathbf{y}_{i,j}, \mathbf{y}_{i',j'}\rangle \mid \mathbf{a} = \hat{\mathbf{a}}] = \begin{cases} o\left(\frac{\|\mathbf{S}\|^2}{\beta(n)^2 k(n)}\right) & \text{if } (i,j) = (i',j'), \\ \\ o\left(\frac{\|\mathbf{S}\|^2}{\beta(n)^2 k(n)} \cdot \frac{1}{2^{k(n)}}\right) & \text{otherwise.} \end{cases} \tag{6.13}$$

Using (6.13) in the expression for $\text{Exp}[\|\mathbf{s}\|^2 \mid \mathbf{a} = \hat{\mathbf{a}}, \mathbf{z} = \hat{\mathbf{z}}]$, we get

$$\begin{aligned} \text{Exp}[\|\mathbf{s}\|^2 \mid \mathbf{a} = \hat{\mathbf{a}}, \mathbf{z} = \hat{\mathbf{z}}] &\le \left(\sum_{i,j} \hat{z}_i^2 + \sum_{(i,j)\ne(i',j')} \frac{|\hat{z}_i| \cdot |\hat{z}_{i'}|}{2^{k(n)}}\right) \cdot o\left(\frac{\|\mathbf{S}\|^2}{\beta(n)^2 k(n)}\right) \\ &\le \left(k(n) \cdot \|\hat{\mathbf{z}}\|^2 + \frac{k(n)^2 \cdot (\sqrt{m(n)}\|\hat{\mathbf{z}}\|)^2}{2^{k(n)}}\right) \cdot o\left(\frac{\|\mathbf{S}\|^2}{\beta(n)^2 k(n)}\right) \\ &\le \left(1 + \frac{m(n)k(n)}{2^{k(n)}}\right) \cdot o\left(\frac{\|\mathbf{S}\|^2 \|\hat{\mathbf{z}}\|^2}{\beta(n)^2}\right). \end{aligned}$$

It follows from the concavity of the square root function that

$$\begin{aligned} \text{Exp}[\|\mathbf{s}\| \mid \mathbf{a} = \hat{\mathbf{a}}, \mathbf{z} = \hat{\mathbf{z}}] &= \text{Exp}[\sqrt{\|\mathbf{s}\|^2} \mid \mathbf{a} = \hat{\mathbf{a}}, \mathbf{z} = \hat{\mathbf{z}}] \\ &\le \sqrt{\text{Exp}[\|\mathbf{s}\|^2 \mid \mathbf{a} = \hat{\mathbf{a}}, \mathbf{z} = \hat{\mathbf{z}}]} \\ &\le o\left(\frac{\|\mathbf{S}\| \cdot \|\hat{\mathbf{z}}\|}{\beta(n)}\right) \cdot \sqrt{1 + \frac{m(n)k(n)}{2^{k(n)}}}. \end{aligned}$$

In order to complete the proof of the lemma, we need to prove bound (6.13). The case $(i,j) = (i',j')$ immediately follows from (6.11):

$$\text{Exp}[\langle \mathbf{y}_{i,j}, \mathbf{y}_{i,j}\rangle \mid \mathbf{a} = \hat{\mathbf{a}}] \le \max \|\mathbf{y}_{i,j}\|^2 = o\left(\frac{\|\mathbf{S}\|^2}{\beta(n)^2 k(n)}\right),$$

where the maximum is over all $\mathbf{y}_{i,j}$ in the support of the sampling algorithm. Now assume $i = i'$, but $j \ne j'$, and let us bound $\text{Exp}[\langle \mathbf{y}_{i,j}, \mathbf{y}_{i,j'}\rangle \mid \mathbf{a} = \hat{\mathbf{a}}]$. Notice that $a_h$ is independent from $\mathbf{y}_{i,j}$ and $\mathbf{y}_{i,j'}$ for all $h \ne i$. Therefore,

$$\text{Exp}[\langle \mathbf{y}_{i,j}, \mathbf{y}_{i,j'}\rangle \mid \mathbf{a} = \hat{\mathbf{a}}] = \text{Exp}[\langle \mathbf{y}_{i,j}, \mathbf{y}_{i,j'}\rangle \mid a_i = \hat{a}_i].$$

We want to bound the conditional expectation of $\langle \mathbf{y}_{i,j}, \mathbf{y}_{i,j'}\rangle$ given $a_i = \hat{a}_i$. Notice that vectors $\mathbf{y}_{i,j}$ and $\mathbf{y}_{i,j'}$ are statistically independent (because they come from different runs of the sampling procedure) and symmetrically distributed (by Lemma 6.6). Therefore,

$$\text{Exp}[\langle \mathbf{y}_{i,j}, \mathbf{y}_{i,j'}\rangle] = \langle \text{Exp}[\mathbf{y}_{i,j}], \text{Exp}[\mathbf{y}_{i,j'}]\rangle = \langle \mathbf{0}, \mathbf{0}\rangle = 0. \tag{6.14}$$

Using (6.14), we immediately get

$$|\,\text{Exp}[\langle \mathbf{y}_{i,j}, \mathbf{y}_{i,j'}\rangle \mid a_i = \hat{a}_i]| = |\,\text{Exp}[\langle \mathbf{y}_{i,j}, \mathbf{y}_{i,j'}\rangle \mid a_i = \hat{a}_i] - \text{Exp}[\langle \mathbf{y}_{i,j}, \mathbf{y}_{i,j'}\rangle]|. \tag{6.15}$$

Notice that, by (6.11), for any $\mathbf{y}_{i,j}$ and $\mathbf{y}_{i,j'}$ in the range of the sampling procedure,

$$|\langle \mathbf{y}_{i,j}, \mathbf{y}_{i,j'}\rangle| \le \|\mathbf{y}_{i,j}\| \cdot \|\mathbf{y}_{i,j'}\| = o\left(\frac{\|\mathbf{S}\|^2}{\beta(n)^2 k(n)}\right).$$

Therefore, by Proposition 2.15, the difference (6.15) is at most

$$2 \cdot o\left(\frac{\|\mathbf{S}\|^2}{\beta(n)^2 k(n)}\right) \cdot \Delta((\mathbf{y}_{i,j}, \mathbf{y}_{i,j'} \mid a_i = \hat{a}_i), (\mathbf{y}_{i,j}, \mathbf{y}_{i,j'})),$$

where $(\mathbf{y}_{i,j}, \mathbf{y}_{i,j'} \mid a_i = \hat{a}_i)$ is the conditional distribution of $(\mathbf{y}_{i,j}, \mathbf{y}_{i,j'})$ given $a_i$. In the following lemma, we bound $\Delta((\mathbf{y}_{i,j}, \mathbf{y}_{i,j'} \mid a_i = \hat{a}_i), (\mathbf{y}_{i,j}, \mathbf{y}_{i,j'}))$.

LEMMA 6.14. *The statistical distance between* $(\mathbf{y}_{i,j}, \mathbf{y}_{i,j'} \mid a_i = \hat{a}_i)$ *and* $(\mathbf{y}_{i,j}, \mathbf{y}_{i,j'})$ *is at most*

$$\Delta((\mathbf{y}_{i,j}, \mathbf{y}_{i,j'} \mid a_i = \hat{a}_i), (\mathbf{y}_{i,j}, \mathbf{y}_{i,j'})) \le \frac{5}{2(2^{k(n)} - 1)}.$$

*Proof.* Notice that $(\mathbf{y}_{i,j}, \mathbf{y}_{i,j'}) = (\mathbf{v}_{i,j} - \psi(\mathbf{w}_{i,j}), \mathbf{v}_{i,j'} - \psi(\mathbf{w}_{i,j'}))$ is a randomized function of $(\mathbf{w}_{i,j}, \mathbf{w}_{i,j'})$, where $\mathbf{v}_{i,j}$ and $\mathbf{v}_{i,j'}$ are computed according to the conditional distribution of the sampling algorithm (given $\mathbf{w} = \mathbf{w}_{i,j}$ or $\mathbf{w} = \mathbf{w}_{i,j'}$). Therefore, by Proposition 2.13,

$$\Delta((\mathbf{y}_{i,j}, \mathbf{y}_{i,j'} \mid a_i = \hat{a}_i), (\mathbf{y}_{i,j}, \mathbf{y}_{i,j'})) \le \Delta((\mathbf{w}_{i,j}, \mathbf{w}_{i,j'} \mid a_i = \hat{a}_i), (\mathbf{w}_{i,j}, \mathbf{w}_{i,j'})).$$

By Proposition 2.18, the distance between $(\mathbf{w}_{i,j}, \mathbf{w}_{i,j'} \mid a_i = \hat{a}_i)$ and $(\mathbf{w}_{i,j}, \mathbf{w}_{i,j'})$ is at most

$$\begin{aligned}
&\Delta((\mathbf{w}_{i,j}, \mathbf{w}_{i,j'} \mid a_i = \hat{a}_i), (\mathbf{w}_{i,j}, \mathbf{w}_{i,j'})) \\
&\qquad \le \frac{1}{2} \max_{\hat{\mathbf{w}}, \hat{\mathbf{w}}'} \left| \frac{\Pr\{a_i = \hat{a}_i \mid \mathbf{w}_{i,j} = \hat{\mathbf{w}}, \mathbf{w}_{i,j'} = \hat{\mathbf{w}}'\}}{\Pr\{a_i = \hat{a}_i\}} - 1 \right| \\
&\qquad = \frac{1}{2} \max_{\hat{\mathbf{w}}, \hat{\mathbf{w}}'} \left| \frac{\Pr\{\sum_{h \notin \{j,j'\}} a_{i,h} = \hat{a}_i - [\hat{\mathbf{w}}]_{\mathbf{M}_n} - [\hat{\mathbf{w}}']_{\mathbf{M}_n}\}}{\Pr\{\sum_{h=1}^{k(n)} a_{i,h} = \hat{a}_i\}} - 1 \right|.
\end{aligned}$$

By Proposition 2.17, the probability at the denominator equals $(1/\#G_n)(1+\epsilon)$ for some $|\epsilon| \le 2^{-k(n)}$. Similarly, the probability at the numerator equals $(1/\#G_n)(1+\epsilon')$ for some $|\epsilon'| \le 2^{-(k(n)-2)}$. It follows that

$$\begin{aligned}
\Delta((\mathbf{w}_{i,j}, \mathbf{w}_{i,j'} \mid a_i = \hat{a}_i), (\mathbf{w}_{i,j}, \mathbf{w}_{i,j'})) &\le \frac{1}{2} \left| \frac{(1/\#G_n)(1+\epsilon')}{(1/\#G_n)(1+\epsilon)} - 1 \right| \\
&= \frac{1}{2} \left| \frac{\epsilon' - \epsilon}{1+\epsilon} \right| \\
&\le \frac{1}{2} \frac{|\epsilon'| + |\epsilon|}{1 - |\epsilon|} \\
&\le \frac{5}{2(2^{k(n)} - 1)}. \qquad \square
\end{aligned}$$

Using Lemma 6.14, we get that

$$\operatorname{Exp}[\langle \mathbf{y}_{i,j}, \mathbf{y}_{i,j'} \rangle \mid \mathbf{a} = \hat{\mathbf{a}}] \le 2 \cdot o\left(\frac{\|\mathbf{S}\|^2}{\beta(n)^2 k(n)}\right) \cdot \frac{5}{2(2^{k(n)} - 1)} = o\left(\frac{\|\mathbf{S}\|^2}{\beta(n)^2 k(n)} \cdot \frac{1}{2^{k(n)}}\right),$$

proving (6.13) for the case when $i = i'$ and $j \ne j'$.

The case when $i \ne i'$ is similar. Consider the conditional distribution of $\mathbf{y}_{i,j}, \mathbf{y}_{i',j'}$ given $\mathbf{a} = \hat{\mathbf{a}}$. Notice that $a_h$ is independent from $\mathbf{y}_{i,j}$ and $\mathbf{y}_{i',j'}$ for all $h \notin \{i, i'\}$. Therefore,

$$\operatorname{Exp}[\langle \mathbf{y}_{i,j}, \mathbf{y}_{i',j'} \rangle \mid \mathbf{a} = \hat{\mathbf{a}}] = \operatorname{Exp}[\langle \mathbf{y}_{i,j}, \mathbf{y}_{i',j'} \rangle \mid a_i = \hat{a}_i, a_{i'} = \hat{a}_{i'}].$$

Moreover, $\mathbf{y}_{i,j}$ and $a_i$ are independent from $\mathbf{y}_{i',j'}$ and $a_{i'}$ because they come from different runs of the sampling algorithm. Therefore,

$$\begin{aligned}\mathrm{Exp}[\langle \mathbf{y}_{i,j}, \mathbf{y}_{i',j'}\rangle \mid a_i = \hat{a}_i, a_{i'} = \hat{a}_{i'}] &= \mathrm{Exp}[\langle (\mathbf{y}_{i,j} \mid a_i = \hat{a}_i), (\mathbf{y}_{i',j'} \mid a_{i'} = \hat{a}_{i'})\rangle] \\ &= \langle \mathrm{Exp}[\mathbf{y}_{i,j} \mid a_i = \hat{a}_i], \mathrm{Exp}[\mathbf{y}_{i',j'} \mid a_{i'} = \hat{a}_{i'}]\rangle,\end{aligned}$$

where, as usual, $(\mathbf{y}_{i,j} \mid a_i = \hat{a}_i)$ (resp., $(\mathbf{y}_{i',j'} \mid a_{i'} = \hat{a}_{i'})$) is the conditional distribution of $\mathbf{y}_{i,j}$ given $a_i$ (resp., $\mathbf{y}_{i',j'}$ given $a_{i'}$). Let $\mathbf{y} = \mathrm{Exp}[\mathbf{y}_{i',j'} \mid a_{i'} = \hat{a}_{i'}]$. We know from (6.11) that $\|\mathbf{y}\| = o(\|\mathbf{S}\|/(\beta(n)\sqrt{k(n)}))$. Since $\mathbf{y}_{i,j}$ is symmetrically distributed, $\langle \mathrm{Exp}[\mathbf{y}_{i,j}], \mathbf{y}\rangle = \langle \mathbf{0}, \mathbf{y}\rangle = 0$, and

$$\begin{aligned}\mathrm{Exp}[\langle \mathbf{y}_{i,j}, \mathbf{y}_{i',j'}\rangle \mid a_i = \hat{a}_i, a_{i'} = \hat{a}_{i'}] &= \langle \mathrm{Exp}[\mathbf{y}_{i,j} \mid a_i = \hat{a}_i], \mathbf{y}\rangle \\ &= \langle \mathrm{Exp}[\mathbf{y}_{i,j} \mid a_i = \hat{a}_i], \mathbf{y}\rangle - \langle \mathrm{Exp}[\mathbf{y}_{i,j}], \mathbf{y}\rangle \\ &= \mathrm{Exp}[\langle (\mathbf{y}_{i,j} \mid a_i = \hat{a}_i), \mathbf{y}\rangle] - \mathrm{Exp}[\langle \mathbf{y}_{i,j}, \mathbf{y}\rangle].\end{aligned}$$

Notice that, by (6.11), for all $\mathbf{y}_{i,j}$ in the range of the sampling algorithm

$$|\langle \mathbf{y}_{i,j}, \mathbf{y}\rangle| \le \|\mathbf{y}_{i,j}\| \cdot \|\mathbf{y}\| = o\left(\frac{\|\mathbf{S}\|^2}{\beta(n)^2 k(n)}\right).$$

Therefore, by Proposition 2.15, the difference between $\mathrm{Exp}[\langle (\mathbf{y}_{i,j} \mid a_i = \hat{a}_i), \mathbf{y}\rangle]$ and $\mathrm{Exp}[\langle \mathbf{y}_{i,j}, \mathbf{y}\rangle]$ is at most

$$2 \cdot o\left(\frac{\|\mathbf{S}\|^2}{\beta(n)^2 k(n)}\right) \cdot \Delta((\mathbf{y}_{i,j} \mid a_i = \hat{a}_i), (\mathbf{y}_{i,j})).$$

Since (for any $j'$) vector $\mathbf{y}_{i,j}$ is a function of $(\mathbf{y}_{i,j}, \mathbf{y}_{i,j'})$, by Proposition 2.13 and Lemma 6.14,

$$\Delta((\mathbf{y}_{i,j} \mid a_i = \hat{a}_i), (\mathbf{y}_{i,j})) \le \Delta((\mathbf{y}_{i,j}, \mathbf{y}_{i,j'} \mid a_i = \hat{a}_i), (\mathbf{y}_{i,j}, \mathbf{y}_{i,j'})) \le \frac{5}{2(2^{k(n)} - 1)}.$$

So, also in this case we have

$$\begin{aligned}\mathrm{Exp}[\langle \mathbf{y}_{i,j}, \mathbf{y}_{i',j'}\rangle \mid \mathbf{a} = \hat{\mathbf{a}}] &\le 2 \cdot o\left(\frac{\|\mathbf{S}\|^2}{\beta(n)^2 k(n)}\right) \cdot \frac{5}{2(2^{k(n)} - 1)} \\ &= o\left(\frac{\|\mathbf{S}\|^2}{\beta(n)^2 k(n)} \cdot \frac{1}{2^{k(n)}}\right). \qquad \square\end{aligned}$$

**7. Applications.** In the previous section we proved (Theorems 6.3 and 6.5) that the problem of finding $n$ linearly independent lattice vectors of length not much bigger than the generalized uniform radius (in the worst case) reduces to the problem of finding small integer solutions to random linear equations on the average. In this section we show how this result can be reformulated as a connection between the average-case and worst-case complexity of various lattice approximation problems. As usual, we refer to the average-case problem as the problem of finding a nonzero integer solution to a random linear equation, but we stress that this is equivalent to finding (approximately) shortest vectors in a random lattice.

We also show that our results imply the existence of provably secure cryptographic (collision resistant) hash functions based on the worst-case hardness of lattice approximation problems.

COROLLARY 7.1. *Let $\tau(n) = n^{O(1)}$ be a function such that there exists a family of $\tau(n)$-perfect easily decodable lattices. For every polynomially bounded function $\mu(m) = m^{O(1)}$ and $m(n) = \Omega(n \log n)$, there exists a sequence of groups $\{G_n\}$ of size $\#G_n = n^{O(n)}$ such that the following is true. If there is a probabilistic polynomial time algorithm $\mathcal{F}$ that on input a uniformly chosen random equation $\mathbf{g} \in G_n^{m(n)}$, outputs with nonnegligible probability a nonzero solution $\mathcal{F}(\mathbf{g}) \in \Lambda(\mathbf{g})$ of length within a factor $\mu(m(n))$ from the shortest (or, more generally, within a factor $\mu(m(n))$ from Minkowski's bound* (2.2)*), then there is a probabilistic polynomial time algorithm that on input any rank $n$ lattice basis $\mathbf{B}$ solves, in the worst case and with high probability, any of the following problems, where $\omega(1)$ is an arbitrary superconstant and polynomially bounded function of $n$:*

1. [SIVP] *Find a set $\mathbf{S} \subseteq \mathcal{L}(\mathbf{B})$ of $n$ linearly independent vectors such that*

$$\|\mathbf{S}\| \leq \omega(1) \cdot \mu(m(n)) \cdot n^{1.5} \cdot \tau(n) \cdot \sqrt{m(n) \cdot \log n} \cdot \lambda_n(\mathcal{L}(\mathbf{B})).$$

2. [GAPSVP] *Compute an approximation $\hat{\lambda}_1$ such that*

$$\frac{\lambda_1(\mathcal{L}(\mathbf{B}))}{\omega(1) \cdot \mu(m(n)) \cdot n^2 \cdot \tau(n) \cdot \sqrt{m(n) \cdot \log n}} \leq \hat{\lambda}_1 \leq \lambda_1(\mathcal{L}(\mathbf{B})).$$

3. [GAPCRP] *Compute an approximation $\hat{\rho}$ such that*

$$\rho(\mathcal{L}(\mathbf{B})) \leq \hat{\rho} \leq \omega(1) \cdot \mu(m(n)) \cdot n^{1.5} \cdot \tau(n) \cdot \sqrt{m(n) \cdot \log n} \cdot \rho(\mathcal{L}(\mathbf{B})).$$

4. [GDD] *Given also a target vector $\mathbf{t} \in \operatorname{span}(\mathbf{B})$, find a lattice vector $\mathbf{v} \in \mathcal{L}(\mathbf{B})$ such that*

$$\|\mathbf{v} - \mathbf{t}\| \leq \omega(1) \cdot \mu(m(n)) \cdot n^{1.5} \cdot \tau(n) \cdot \sqrt{m(n) \cdot \log n} \cdot \rho(\mathcal{L}(\mathbf{B})).$$

*Proof.* Let $\beta(n) = \sqrt{\omega(1) \cdot m(n)} \cdot \mu(m(n))$, $\gamma(n) = \beta(n)\tau(n)\sqrt{\omega(1) \cdot \log n}$ and $G_n$ be as defined in Theorem 6.5. Notice that $\gamma(n) \leq \omega(1) \cdot \sqrt{m(n) \cdot \log n} \cdot \tau(n) \cdot \mu(m(n)) = n^{O(1)}$ and $\#G_n \leq (n^{1.5}\gamma(n)/8)^n = n^{O(n)}$. Therefore, by Theorem 5.5, any equation $\mathbf{g} \in G_n^{m(n)}$ has a nonzero solution of length at most $O(\sqrt{m(n)})$. Let $\mathcal{F}(\cdot)$ be a probabilistic polynomial time algorithm to find nonzero solutions of length within a factor $\mu(m(n))$ from the shortest (with nonnegligible probability). We know that, when successful, $\mathcal{F}(\cdot)$ finds solutions of length at most

$$\|\mathcal{F}(\mathbf{g})\| \leq \mu(m(n)) O(\sqrt{m(n)}) \leq \beta(n).$$

So, algorithm $\mathcal{F}(\cdot)$ solves $\mathrm{HSIS}_{G,m,\beta}$ on the average with nonnegligible probability. Combining $\mathcal{F}$ with the reductions from Theorems 6.5 and 6.3, we get a polynomial time algorithm $\mathcal{S}(\cdot)$ that solves $\mathrm{GIVP}_\gamma^{\hat{\zeta}}$ (in the worst case and with high probability) for approximation factor

$$\gamma(n) = \beta(n) \cdot \tau(n) \cdot \sqrt{\omega(1) \cdot \log n} = \omega(1) \cdot \sqrt{m(n) \cdot \log n} \cdot \mu(m(n)) \cdot \tau(n).$$

We show how to use this algorithm to solve all the worst-case problems in the conclusion of the corollary.

1. [SIVP] Just run $\mathbf{S} = \mathcal{S}(\mathbf{B})$ and output $\mathbf{S}$. By (3.1),

$$\begin{aligned}\|\mathbf{S}\| \leq \gamma(n) \cdot \hat{\zeta}(\mathcal{L}(\mathbf{B})) &\leq \frac{3}{2} n^{1.5} \cdot \gamma(n) \cdot \lambda_n(\mathcal{L}(\mathbf{B})) \\ &= \frac{3}{2} \omega(1) \cdot n^{1.5} \cdot \tau(n) \cdot \mu(m(n)) \cdot \sqrt{m(n) \cdot \log n} \cdot \lambda_n(\mathcal{L}(\mathbf{B})).\end{aligned}$$

2. [GapSVP] On input basis $\mathbf{B}$, run $\mathbf{S} = \mathcal{S}(\mathbf{B}^*)$, where $\mathbf{B}^*$ is the basis of the dual lattice, and output $1/\|\mathbf{S}\|$. By (2.5),

$$\|\mathbf{S}\| \geq \lambda_n(\mathcal{L}(\mathbf{B})^*) \geq \frac{1}{\lambda_1(\mathcal{L}(\mathbf{B}))}.$$

Also, by (3.2),

$$\begin{aligned}\|\mathbf{S}\| \leq \gamma(n) \cdot \hat{\zeta}(\mathcal{L}(\mathbf{B})^*) &\leq \frac{3}{2}n^2 \cdot \gamma(n)/\lambda_1(\mathcal{L}(\mathbf{B})) \\ &= \frac{3}{2}\omega(1) \cdot n^2 \cdot \tau(n) \cdot \mu(m(n)) \cdot \sqrt{m(n) \cdot \log n}/\lambda_1(\mathcal{L}(\mathbf{B})).\end{aligned}$$

3. [GapCRP] This time, we run $\mathbf{S} = \mathcal{S}(\mathbf{B})$ and output $\sqrt{n}\|\mathbf{S}\|/2$. By (2.3),

$$\sqrt{n}\|\mathbf{S}\|/2 \geq (\sqrt{n}/2)\lambda_n(\mathcal{L}(\mathbf{B})) \geq \rho(\mathcal{L}(\mathbf{B})).$$

Moreover, by Theorem 3.6,

$$\sqrt{n}\|\mathbf{S}\|/2 \leq \frac{3}{2}n^{1.5}\cdot\gamma(n)\cdot\rho(\mathcal{L}(\mathbf{B})) = \frac{3}{2}\omega(1)\cdot n^{1.5}\cdot\tau(n)\cdot\mu(m(n))\cdot\sqrt{m(n)\cdot\log n}\cdot\rho(\mathcal{L}(\mathbf{B})).$$

4. [GDD] In order to find a lattice point close to target $\mathbf{t}$, we first run $\mathbf{S} = \mathcal{S}(\mathbf{B})$ and then execute Babai's nearest plane algorithm [6] using sublattice $\mathbf{S}$ and target $\mathbf{t}$. The result is a point within distance $\sqrt{n}\|\mathbf{S}\|/2$ from the target. As in the proof for the CRP, this bound satisfies

$$\sqrt{n}\|\mathbf{S}\|/2 \leq \frac{3}{2}\omega(1) \cdot n^{1.5} \cdot \tau(n) \cdot \mu(m(n)) \cdot \sqrt{m(n) \cdot \log n} \cdot \rho(\mathcal{L}(\mathbf{B})). \qquad \square$$

Notice that in the proof of Corollary 7.1, the definition of group $G_n$ implicitly depends on the function $m(n)$. This is because in Theorem 6.5 the definition of group $G_n$ depends on the value of $\alpha(n)$, which in turn depends (via $\gamma(n)$) on the value of $\beta(n)$. Moreover, the definition of $\beta(n)$ in the proof of Corollary 7.1 depends on $\mu(m(n))$. So, unless $\mu(\cdot)$ is a constant function, group $G_n$ can be selected only after the value of $m(n)$ has been chosen. The following corollary immediately follows from Corollary 7.1 by setting $m(n) = \Theta(n \log n)$ and $\mu(m) = 1$ and observing that the definition of group $G_n$ does not depend on $m(n)$ when $\mu(m)$ is constant.

Corollary 7.2. *Let $\tau(n) = n^{O(1)}$ be a function such that there exists a family of $\tau(n)$-perfect easily decodable lattices. For every superlogarithmic function $\omega(\log n)$, there exists a sequence of groups $\{G_n\}$ of size $\#G_n = n^{O(n)}$ such that for any $m(n) = \Theta(n \log n)$, the following is true. If there is a probabilistic polynomial time algorithm $\mathcal{F}(\cdot)$ that on input a uniformly chosen random equation $\mathbf{g} \in G_n^{m(n)}$, outputs with non-negligible probability a shortest nonzero solution $\mathcal{F}(\mathbf{g}) \in \Lambda(\mathbf{g})$ (or, more generally, a solution satisfying Minkowski's bound* (2.2) $\|\mathcal{F}(\mathbf{g})\| \leq \sqrt{m(n)}\cdot\det(\Lambda(\mathbf{g}))^{1/m(n)}$*), then there is a probabilistic polynomial time algorithm that on input any rank n lattice basis* $\mathbf{B}$ *solves, in the worst case and with high probability, any of the following problems:*

1. [SIVP] *Find a maximal set of linearly independent vectors of length within* $n^2 \cdot \tau(n) \cdot \omega(\log n)$ *from the shortest.*
2. [GapSVP] *Approximate* $\lambda_1(\mathcal{L}(\mathbf{B}))$ *within a factor* $n^{2.5} \cdot \tau(n) \cdot \omega(\log n)$.
3. [GapCRP] *Approximate* $\rho(\mathcal{L}(\mathbf{B}))$ *within a factor* $n^2 \cdot \tau(n) \cdot \omega(\log n)$.
4. [GDD] *Given also a target vector* $\mathbf{t} \in \operatorname{span}(\mathbf{B})$, *find a lattice vector* $\mathbf{v} \in \mathcal{L}(\mathbf{B})$ *within distance* $n^2 \cdot \tau(n) \cdot \omega(\log n) \cdot \rho(\mathcal{L}(\mathbf{B}))$ *from* $\mathbf{t}$.

We now turn to the construction of collision resistant hash functions. Following [18], for any $\mathbf{g} \in G_n^{m(n)}$, define function $h_{\mathbf{g}}\colon \{0,1\}^{m(n)} \to G_n$ by

$$h_{\mathbf{g}}(\mathbf{x}) = \sum_{i=1}^{n} g_i x_i.$$

Notice that function $h_{\mathbf{g}}$ maps $m(n)$ bits to $\log_2 \#G$ bits. If $m(n) > \log_2 \#G$, then the function compresses the input $\mathbf{x}$, and collisions $h_{\mathbf{g}}(\mathbf{x}) = h_{\mathbf{g}}(\mathbf{y})$ (for $\mathbf{x} \neq \mathbf{y}$) are guaranteed to exist by the pigeon hole principle. We prove that, if the key $\mathbf{g}$ is chosen at random, then these collisions are computationally hard to find.

COROLLARY 7.3. *Let $\tau(n) = n^{O(1)}$ be a function such that there exists a family of $\tau(n)$-perfect easily decodable lattices. For every superlogarithmic function $\omega(\log n)$, there exists a sequence of groups $\{G_n\}$ of size $\#G_n = n^{O(n)}$ such that the following is true. Assume no probabilistic polynomial time algorithm can solve problems* SIVP, GAPSVP, GAPCRP, *or* GDD *(in the worst case and with high probability) within the factors specified in Corollary 7.2. Then for any $c > 1$ and $m(n) = \max\{c \cdot \log_2 \#G_n, \Theta(n \log n)\}$, there exists no probabilistic polynomial time algorithm that on input a random key $\mathbf{g} \in G_n^{m(n)}$ outputs with nonnegligible probability an $h_{\mathbf{g}}$-collision, i.e., two binary vectors $\mathbf{x} \neq \mathbf{y}$ such that $h_{\mathbf{g}}(\mathbf{x}) = h_{\mathbf{g}}(\mathbf{y})$.*

*Proof.* Notice that $m(n) \geq c \cdot \log_2 \#G_n$, so function $h_{\mathbf{g}}$ is a hash function with compression ratio $c$. Assume, for contradiction, that $\mathcal{F}(\mathbf{g}) = (\mathbf{x}, \mathbf{y})$ is a collision finder algorithm with nonnegligible success probability, and notice that if $\mathcal{F}$ is successful, then $\mathbf{x} - \mathbf{y} \in \Lambda(\mathbf{g}) \setminus \{\mathbf{0}\}$ is a nonzero solution to equation $\mathbf{g}$ of length at most

$$\|\mathbf{x} - \mathbf{y}\| \leq \sqrt{m(n)}.$$

Since $\Lambda(\mathbf{g})$ is a sublattice of $\mathbb{Z}^n$, $\det(\Lambda(\mathbf{g})) \geq \det(\mathbb{Z}^n) = 1$, and solution $\mathbf{x} - \mathbf{y} \in \Lambda(\mathbf{g})$ satisfies Minkowski's bound (2.2)

$$\|\mathbf{x} - \mathbf{y}\| \leq \sqrt{m(n)} \leq \sqrt{m(n)}\det(\Lambda(\mathbf{g}))^{1/m(n)}.$$

In order to apply Corollary 7.2 and get a contradiction, we need only to show that $m(n) = \Theta(n \log n)$. The lower bound $m(n) = \Omega(n \log n)$ immediately follows from the definition of $m(n) \geq \Theta(n \log n)$. The upper bound $m(n) = O(n \log n)$ follows from the fact that $\#G_n = n^{O(n)}$. This proves that $m(n) = \Theta(n \log n)$, and by Corollary 7.2 there exist probabilistic polynomial time algorithms to approximately solve SIVP, GAPSVP, GAPCRP, and GDD in the worst case and with high probability. ∎

We conclude the section with two remarks about the choice of the groups $G_n$ in the previous corollaries.

*Remark.* It can be shown that the groups $G_n$ defined in the proofs of Corollaries 7.1, 7.2, and 7.3 have size $\#G_n = n^{\Theta(n)}$. In particular, in Corollary 7.3, we could have simply defined $m(n) = c \log_2 \#G_n$, instead of $\max\{c \log_2 \#G_n, \Theta(n \log n)\}$, because $c \log_2 \#G_n = \Theta(n \cdot \log n)$.

*Remark.* Corollaries 7.1, 7.2, and 7.3 are pretty flexible in terms of the choice of group $G_n$. The only property required for the proof to go through is that $G_n$ is a group of size $n^{\Theta(n)}$ that can be represented as the quotient of an easily decodable $\tau(n)$-perfect lattice $\mathcal{L}(\mathbf{L}_n)$ modulo a sublattice $\mathcal{L}(\mathbf{M}_n)$ such that (6.2) holds true.

**8. Conclusion and open problems.** We related the computational complexity of finding (approximately) shortest nonzero integer solutions to random linear equations with coefficients in a suitably chosen group (on the average and with non-negligible probability) to the worst-case complexity of approximating various lattice problems. Since the set of integer solutions to a homogeneous linear equation forms a lattice, the result can be interpreted as a connection between the average-case and worst-case complexity of various lattice problems. The connection immediately also gives provably secure cryptographic hash functions that are as hard to break on the average as the worst-case complexity of approximating various lattice problems within polynomial factors. The worst-case approximation factors achieved depend on the class of easily decodable lattices used in the definition of the class of equations (or cryptographic hash functions). In particular, if $\tau(n)$-perfect easily decodable lattices are used, then finding shortest solutions to random equations (or finding collisions to hash functions) is at least as hard as approximating the length of the shortest vector in any lattice, in the worst case, within a factor $\gamma(n) = n^{2.5}\tau(n)\omega(\log n)$. Even for $\tau(n) = \sqrt{n}$ (which corresponds, as a special case, to Ajtai's random class of equations), this improves the previously known best connection factor of [11] by more than $O(n)$. We also showed that finding shortest solutions to random equations is at least as hard as approximating within a factor $n^2\tau(n)\omega(\log n)$ any of the following problems:

(i) [SIVP] computing a maximal set of shortest linearly independent vectors,
(ii) [GapCRP] computing the covering radius, and
(iii) [GDD] computing a lattice vector within distance $\max_{\mathbf{x}} \operatorname{dist}(\mathbf{x}, \mathcal{L}(\mathbf{B}))$ from a given target,

improving [11] in the case of SIVP by more than $O(\sqrt{n})$, and connecting the average case complexity of solving random equations to two new computational problems on lattices that might be of independent interest.

We also gave polynomial time constructions of easily decodable $\tau(n)$-perfect lattices with $\tau(n) = o(\sqrt{n})$. These constructions allow us to achieve approximation factors $n^{2.5}\omega(\sqrt{\log n \log\log n})$ (for SIVP, GapCRP, and GDD) and $n^3\omega(\sqrt{\log n \log\log n})$ (for SVP). While this improvement over $\tau(n) = \sqrt{n}$ is not substantial, it suggests that further investigation of almost perfect lattices might allow us to find easily decodable $\tau(n)$-perfect lattices with much smaller $\tau(n)$, e.g., $\tau(n) = n^\epsilon$ or even $\tau(n) = O(1)$. This would immediately reduce the approximation factor for all the above problems by about $\sqrt{n}$.

Another possible source of improvement are better bounds relating the fundamental constants associated to a lattice. Our main theorem (Theorem 6.5) shows that finding short solutions on the average is at least as hard as finding vectors that are not much longer than a new lattice quantity called the generalized uniform radius. All other results are obtained by first relating the generalized uniform radius to the covering radius (Theorem 3.6), and then bounding the covering radius in terms of other lattice constants using standard transference theorems and other well-known bounds (Proposition 2.8). In particular, (3.1) and (3.2) show that the generalized uniform radius $\hat{\zeta}(\mathcal{L}(\mathbf{B}))$ is at most $O(n^{1.5})$ times $\lambda_n(\mathcal{L}(\mathbf{B}))$ or at most $O(n^2)$ times $1/\lambda_1(\mathcal{L}(\mathbf{B})^*)$. It would be interesting to improve (3.1) and (3.2) to show, for example, that

$$\hat{\zeta}(\mathcal{L}(\mathbf{B})) \leq O(n)\lambda_n(\mathcal{L}(\mathbf{B})) \tag{8.1}$$

and

$$\hat{\zeta}(\mathcal{L}(\mathbf{B})) \leq O(n)/\lambda_1(\mathcal{L}(\mathbf{B})^*). \tag{8.2}$$

Whether these bounds hold true is a natural geometric question, and proving them would be of independent interest. Moreover, it would allow us to reduce the approximation factors for SIVP and SVP by $O(\sqrt{n})$ and $O(n)$, respectively. Together with the construction of better almost perfect easily decodable lattices, this would immediately improve the approximation factors for both SVP and SIVP to just $n^{1.5}\omega(\log n)$. Connections with such small approximation factors are currently known only for restrictions of the (worst-case) SVP to lattices with special structure where the shortest vector is unique in some technical sense [39].

Notice that by (2.5), bound (8.2) would also imply (8.1). Also, (8.2), if correct, would be asymptotically optimal because Conway and Thompson (see [37]) showed that there exist self-dual lattices such that $\rho(\mathcal{L}(\mathbf{B})) \cdot \lambda_1(\mathcal{L}(\mathbf{B})^*) \geq O(n)$, and by Proposition 3.2 $\rho(\mathcal{L}(\mathbf{B})) \leq \zeta(\mathcal{L}(\mathbf{B})) \leq \hat{\zeta}(\mathcal{L}(\mathbf{B}))$. We conjecture that (8.2) holds true and that there exist classes of random equations such that finding shortest nonzero solutions on the average (with nonnegligible probability) is at least as hard as approximating the length of the shortest nonzero vector (or finding a maximal set of shortest linearly independent vectors) in any $n$-dimensional lattice within a factor $n^{1.5}\omega(\log n)$.

**Acknowledgments.** The author would like to thank Ravi Kannan and Alex Vardy for interesting discussions and pointers to relevant references, Oded Regev and an anonymous reviewer for their many and valuable comments that helped to simplify and improve the presentation, and Oded Goldreich for his invaluable feedback about the paper.

## REFERENCES

[1] D. Aharonov and O. Regev, *Lattice problems in NP intersect coNP*, in 45th Annual IEEE Symposium on Foundations of Computer Science, Rome, Italy, 2004, to appear.

[2] M. Ajtai, *Generating hard instances of lattice problems (extended abstract)*, in Proceedings of the 28th Annual ACM Symposium on Theory of Computing, Philadelphia, PA, 1996, pp. 99–108.

[3] M. Ajtai, *The shortest vector problem in $l_2$ is NP-hard for randomized reductions (extended abstract)*, in Proceedings of the 30th Annual ACM Symposium on Theory of Computing, Dallas, TX, 1998, pp. 10–19.

[4] M. Ajtai and C. Dwork, *A public-key cryptosystem with worst-case/average-case equivalence*, in Proceedings of the 29th Annual ACM Symposium on Theory of Computing, El Paso, TX, 1997, pp. 284–293.

[5] M. Ajtai, R. Kumar, and D. Sivakumar, *A sieve algorithm for the shortest lattice vector problem*, in Proceedings of the 33rd Annual ACM Symposium on Theory of Computing, Heraklion, Crete, Greece, 2001, pp. 266–275.

[6] L. Babai, *On Lovasz' lattice reduction and the nearest lattice point problem*, Combinatorica, 6 (1986), pp. 1–13.

[7] W. Banaszczyk, *New bounds in some transference theorems in the geometry of numbers*, Math. Ann., 296 (1993), pp. 625–635.

[8] J. Blömer and J.-P. Seifert, *On the complexity of computing short linearly independent vectors and short bases in a lattice*, in Proceedings of the 31st Annual ACM Symposium on Theory of Computing, Atlanta, GA, 1999, pp. 711–720.

[9] J. Bruck and M. Naor, *The hardness of decoding linear codes with preprocessing*, IEEE Trans. Inform. Theory, 36 (1990), pp. 381–385.

[10] G. J. Butler, *Simultaneous packing and covering in Euclidean space*, Proc. London Math. Soc., 25 (1972), pp. 721–735.

[11] J.-Y. Cai and A. P. Nerurkar, *An improved worst-case to average-case connection for lattice problems (extended abstract)*, in Proceedings of the 38th Annual IEEE Symposium on Foundations of Computer Science, Miami Beach, FL, 1997, pp. 468–477.

[12] J. H. Conway and N. J. A. Sloane, *Sphere Packings, Lattices and Groups*, 3rd ed., Springer-Verlag, New York, 1998.

[13] I. Dinur, G. Kindler, R. Raz, and S. Safra, *Approximating CVP to within almost-polynomial factors is NP-hard*, Combinatorica, 23 (2003), pp. 205–243.

[14] M. Dyer, A. Frieze, and R. Kannan, *A random polynomial-time algorithm for approximating the volume of convex bodies*, J. ACM, 38 (1991), pp. 1–17.

[15] U. Feige and D. Micciancio, *The inapproximability of lattice and coding problems with preprocessing*, J. Comput. System Sci., 69 (2004), pp. 45–67.

[16] O. Goldreich, *Foundation of Cryptography—Basic Tools*, Cambridge University Press, Cambridge, UK, 2001.

[17] O. Goldreich and S. Goldwasser, *On the limits of nonapproximability of lattice problems*, J. Comput. System Sci., 60 (2000), pp. 540–563.

[18] O. Goldreich, S. Goldwasser, and S. Halevi, *Collision-free hashing from lattice problems*, Tech. report TR96-056, Electronic Colloquium on Computational Complexity, 1996, http://www.eccc.uni-trier.de/eccc/.

[19] O. Goldreich, D. Micciancio, S. Safra, and J.-P. Seifert, *Approximating shortest lattice vectors is not harder than approximating closest lattice vectors*, Inform. Process. Lett., 71 (1999), pp. 55–61.

[20] V. Guruswami, D. Micciancio, and O. Regev, *The complexity of the covering radius problem on lattices and codes*, in Proceedings of the 19th Annual IEEE Conference on Computational Complexity, Amherst, MA, 2004, pp. 161–173.

[21] I. Honkala and A. Tietäväinen, *Codes and number theory*, Handbook of Coding Theory, Vol. 2, Elsevier, New York, 1998, pp. 1141–1194.

[22] R. Kannan, *Algorithmic geometry of numbers*, in Annual Reviews of Computer Science Vol. 2, Annual Review Inc., Palo Alto, CA, 1987, pp. 231–267.

[23] R. Kannan, *Minkowski's convex body theorem and integer programming*, Math. Oper. Res., 12 (1987), pp. 415–440.

[24] R. Kannan, *Lattice translates of a polytope and the Frobenius problem*, Combinatorica, 12 (1992), pp. 161–177.

[25] R. Kannan and S. Vempala, *Sampling lattice points*, in Proceedings of the 29th Annual ACM Symposium on Theory of Computing, El Paso, TX, 1997, pp. 696–700.

[26] J. C. Lagarias, H. W. Lenstra, Jr., and C.-P. Schnorr, *Korkine-Zolotarev bases and successive minima of a lattice and its reciprocal lattice*, Combinatorica, 10 (1990), pp. 333–348.

[27] A. K. Lenstra, H. W. Lenstra, Jr., and L. Lovász, *Factoring polynomials with rational coefficients*, Math. Ann., 261 (1982), pp. 513–534.

[28] A. Lobstein, *The hardness of solving subset sum with preprocessing*, IEEE Trans. Inform. Theory, 36 (1990), pp. 943–946.

[29] J. E. Mazo and A. M. Odlyzko, *Lattice points in high dimensional spheres*, Monatsh. Math., 110 (1990), pp. 47–61.

[30] A. McLoughlin, *The complexity of computing the covering radius of a code*, IEEE Trans. Inform. Theory, 30 (1984), pp. 800–804.

[31] D. Micciancio, *The hardness of the closest vector problem with preprocessing*, IEEE Trans. Inform. Theory, 47 (2001), pp. 1212–1215.

[32] D. Micciancio, *Improving lattice based cryptosystems using the Hermite normal form*, in Cryptography and Lattices Conference, Lecture Notes in Comput. Sci. 2146, J. Silverman, ed., Springer-Verlag, Berlin, 2001, pp. 126–145.

[33] D. Micciancio, *The shortest vector in a lattice is hard to approximate to within some constant*, SIAM J. Comput., 30 (2001), pp. 2008–2035.

[34] D. Micciancio, *Generalized compact knapsaks, cyclic lattices, and efficient one-way functions from worst-case complexity assumptions*, in Proceedings of the 43rd Annual Symposium on Foundations of Computer Science, Vancouver, BC, Canada, 2002, pp. 356–365.

[35] D. Micciancio, *Improved cryptographic hash functions with worst-case/average-case connection*, in Proceedings of the 34th Annual ACM Symposium on Theory of Computing, Montréal, Québec, Canada, 2002, pp. 609–618.

[36] D. Micciancio and S. Goldwasser, *Complexity of Lattice Problems: A Cryptographic Perspective*, Kluwer Internat. Ser. Engrg. Comput. Sci. 671, Kluwer Academic Publishers, Boston, 2002.

[37] J. Milnor and D. Husemoller, *Symmetric Bilinear Forms*, Springer-Verlag, New York, 1973.

[38] O. Regev, *Improved inapproximability of lattice and coding problems with preprocessing*, in Proceedings of the 18th Annual IEEE Conference on Computational Complexity, Århus, Denmark, 2003, pp. 315 –322.

[39] O. Regev, *New lattice based cryptographic constructions*, in Proceedings of the 35th Annual ACM Symposium on Theory of Computing, San Diego, CA, 2003, pp. 407–426.

[40] C. A. Rogers, *A note on coverings and packings*, J. London Math. Soc., 25 (1950), pp. 327–331.

[41] C.-P. Schnorr, *A hierarchy of polynomial time lattice basis reduction algorithms*, Theoret. Comput. Sci., 53 (1987), pp. 201–224.

[42] A. Vardy, *Algorithmic complexity in coding theory and the minimum distance problem*, in Proceedings of the 29th Annual ACM Symposium on Theory of Computing, El Paso, TX, 1997, pp. 92–109.

SIAM J. COMPUT. © 2004 Society for Industrial and Applied Mathematics
Vol. 34, No. 1, pp. 170–194

# SMALL SPANS IN SCALED DIMENSION*

JOHN M. HITCHCOCK†

**Abstract.** Juedes and Lutz [*SIAM J. Comput.*, 24 (1995), pp. 279–295] proved a *small span theorem* for polynomial-time many-one reductions in exponential time. This result says that for language $A$ decidable in exponential time, either the class of languages reducible to $A$ (the lower span) or the class of problems to which $A$ can be reduced (the upper span) is small in the sense of resource-bounded measure and, in particular, that the degree of $A$ is small. Small span theorems have been proved for increasingly stronger polynomial-time reductions, and a small span theorem for polynomial-time Turing reductions would imply BPP $\neq$ EXP. In contrast to the progress in resource-bounded measure, Ambos-Spies et al. [*Proceedings of the* 16*th IEEE Conference on Computational Complexity*, Philadelphia, PA, IEEE Computer Society, Los Alamitos, CA, 2001, pp. 210–217] showed that there is no small span theorem for the resource-bounded dimension of Lutz [*SIAM J. Comput.*, 32 (2003), pp. 1236–1259], even for polynomial-time many-one reductions.

Resource-bounded scaled dimension, recently introduced by Hitchcock, Lutz, and Mayordomo [*J. Comput. System Sci.*, 69 (2004), pp. 97–122], provides rescalings of resource-bounded dimension. We use scaled dimension to further understand the contrast between measure and dimension regarding polynomial-time spans and degrees. We strengthen prior results by showing that the small span theorem holds for polynomial-time many-one reductions in the $-3$rd-order scaled dimension, but fails to hold in the $-2$nd-order scaled dimension. Our results also hold in exponential space.

As an application, we show that determining the $-2$nd- or $-1$st-order scaled dimension in ESPACE of the many-one complete languages for E would yield a proof of P = BPP or P $\neq$ PSPACE. On the other hand, it is shown unconditionally that the complete languages for E have $-3$rd-order scaled dimension 0 in ESPACE and $-2$nd- and $-1$st-order scaled dimension 1 in E.

**Key words.** resource-bounded dimension, small span theorems, polynomial-time degrees

**AMS subject classification.** 68Q15

**DOI.** 10.1137/S0097539703426416

## 1. Introduction.

Resource-bounded measure [16] defines the relative size of classes of decision problems and has been used very successfully to study polynomial-time reductions within exponential-time complexity classes. Measure-theoretic arguments were the first to show that for all $\alpha < 1$, every $\leq^{\mathrm{p}}_{n^\alpha-\mathrm{tt}}$-hard language for exponential time is exponentially dense [19]. The first plausible hypothesis on NP to separate the $\leq^{\mathrm{p}}_{\mathrm{m}}$ and $\leq^{\mathrm{p}}_{\mathrm{T}}$ reducibilities within NP came from resource-bounded measure [20].

The *degrees* and *spans* of languages under polynomial-time reductions have also been studied by several researchers using resource-bounded measure. For a reducibility $\leq^{\mathrm{p}}_r$ and any $A \subseteq \{0,1\}^*$, the $\leq^{\mathrm{p}}_r$-*lower span of* $A$ is the class $\mathrm{P}_r(A)$ of all languages that are $\leq^{\mathrm{p}}_r$-reducible to $A$, the $\leq^{\mathrm{p}}_r$-*upper span of* $A$ is the class $\mathrm{P}_r^{-1}(A)$ of all languages to which $A$ is $\leq^{\mathrm{p}}_r$-reducible, and the $\leq^{\mathrm{p}}_r$-*degree of* $A$ is the class $\deg^{\mathrm{p}}_r(A) = \mathrm{P}_r(A) \cap \mathrm{P}_r^{-1}(A)$. Juedes and Lutz [12] proved the following *small span theorem* for $\leq^{\mathrm{p}}_{\mathrm{m}}$-reductions in both E and in EXP. Here the notation $\mu(\mathcal{C} \mid \mathcal{D})$ denotes the *measure of* $\mathcal{C}$ *within* $\mathcal{D}$, where $\mathcal{D}$ is a suitable complexity class. If $\mu(\mathcal{C} \mid \mathcal{D}) = 0$, then intuitively $\mathcal{C} \cap \mathcal{D}$ is a negligible subset of $\mathcal{D}$.

*Received by the editors April 23, 2003; accepted for publication (in revised form) April 17, 2004; published electronically October 28, 2004. This research was supported in part by National Science Foundation grant 9988483.

http://www.siam.org/journals/sicomp/34-1/42641.html

†Department of Computer Science, University of Wyoming, Laramie, WY 82071-3315 (jhitchco@cs.uwyo.edu).

THEOREM 1.1 (Juedes and Lutz [12]). *Let $\mathcal{D} \in \{\mathrm{E}, \mathrm{EXP}\}$. For every $A \in \mathcal{D}$,*

$$\mu(\mathrm{P_m}(A) \mid \mathcal{D}) = 0$$

*or*

$$\mu(\mathrm{P_m^{-1}}(A) \mid \mathcal{D}) = 0.$$

*In particular,* $\mu(\deg^{\mathrm{p}}_{\mathrm{m}}(A) \mid \mathcal{D}) = 0$.

That is, at least one of the upper or lower spans of $A$ is small within $\mathcal{D}$. Using a result of Bennett and Gill [4], Juedes and Lutz [12] noted that strengthening Theorem 1.1 from $\leq^{\mathrm{p}}_{\mathrm{m}}$-reductions to $\leq^{\mathrm{p}}_{\mathrm{T}}$-reductions would achieve the separation BPP $\neq$ EXP. Pursuing this program, small span theorems for reductions of progressively increasing strength between $\leq^{\mathrm{p}}_{\mathrm{m}}$ and $\leq^{\mathrm{p}}_{\mathrm{T}}$ have been obtained by Lindner [14], Ambos-Spies, Neis, and Terwijn [3], and Buhrman and van Melkebeek [6].

Resource-bounded dimension was introduced by Lutz [18] as an effectivization of Hausdorff dimension [9] to investigate the fractal structure of complexity classes. Just like resource-bounded measure, resource-bounded dimension is defined within suitable complexity classes $\mathcal{D}$. For any complexity class $\mathcal{C}$, the *dimension of $\mathcal{C}$ within $\mathcal{D}$* is a real number in $[0, 1]$ and is denoted by $\dim(\mathcal{C} \mid \mathcal{D})$. If $\dim(\mathcal{C} \mid \mathcal{D}) < 1$, then $\mu(\mathcal{C} \mid \mathcal{D}) = 0$, but the converse may fail. This means that resource-bounded dimension is capable of quantitatively distinguishing among the measure 0 sets. With regard to the measure 0 sets in Theorem 1.1, Ambos-Spies, Merkle, Reimann, and Stephan [2] proved the following.

THEOREM 1.2 (Ambos-Spies et al. [2]). *For every $A \in \mathrm{E}$,*

$$\dim(\deg^{\mathrm{p}}_{\mathrm{m}}(A) \mid \mathrm{E}) = \dim(\mathrm{P_m}(A) \mid \mathrm{E}).$$

In particular, as $\dim(\mathrm{E} \mid \mathrm{E}) = 1$, the $\leq^{\mathrm{p}}_{\mathrm{m}}$-complete degree for E has dimension 1 within E. This implies that replacing "$\mu$" by "dim" in Theorem 1.1 makes the statement for E no longer true. In other words, there is no analogue of the small span theorem for dimension in E. Dimension in E cannot distinguish between lower spans and degrees.

To overcome limitations of resource-bounded dimension for investigating complexity classes within ESPACE, Hitchcock, Lutz, and Mayordomo [11] introduced for each integer $i \in \mathbb{Z}$ an *ith-order scaled dimension* $\dim^{(i)}(\cdot \mid \mathcal{D})$. For any class $\mathcal{C}$ and $i \in \mathbb{Z}$, $\dim^{(i)}(\mathcal{C} \mid \mathcal{D}) \in [0, 1]$, and if it is less than 1, then $\mu(\mathcal{C} \mid \mathcal{D}) = 0$. The quantity $\dim^{(i)}(\mathcal{C} \mid \mathcal{D})$ is nondecreasing in $i$, and there is at most one $i \in \mathbb{Z}$ for which $0 < \dim^{(i)}(\mathcal{C} \mid \mathcal{D}) < 1$. The 0th-order dimension, $\dim^{(0)}(\cdot \mid \mathcal{D})$, is precisely the standard unscaled dimension, and the other orders can be more useful than it for certain complexity classes. To illustrate this, we mention some examples from circuit-size complexity. For a function $s : \mathbb{N} \to \mathbb{N}$, let $\mathrm{SIZE}(s(n))$ consist of all languages decidable by nonuniform Boolean circuit families of size at most $s(n)$. Lutz [18] showed that

$$\dim\left(\mathrm{SIZE}\left(\alpha\frac{2^n}{n}\right)\middle|\,\mathrm{ESPACE}\right) = \alpha \tag{1.1}$$

for all $\alpha \in (0, 1)$. Circuit size bounds of the form $2^{\alpha n}$ and $2^{n^\alpha}$ are typically of more interest in complexity theory, but (1.1) implies that $\mathrm{SIZE}(2^{\alpha n})$ and $\mathrm{SIZE}(2^{n^\alpha})$ have dimension 0 in ESPACE for all $\alpha \in (0,1)$. For these size bounds, the scaled dimensions are useful; in [11] it is shown that

$$\dim^{(1)}(\mathrm{SIZE}(2^{\alpha n}) \mid \mathrm{ESPACE}) = \alpha$$

and

$$\dim^{(2)}(\mathrm{SIZE}(2^{n^\alpha}) \mid \mathrm{ESPACE}) = \alpha$$

for any $\alpha \in (0,1)$.

This paper uses scaled dimension to investigate polynomial-time spans and degrees and further understand the contrast between Theorems 1.1 and 1.2. We show that the same dichotomy also occurs between the −3rd- and −2nd-orders of scaled dimension. The main contribution of this paper is a strengthening of Theorem 1.1 to give a small span theorem for scaled dimension. (The following is a corollary of a stronger result proved in Theorem 6.3.)

Theorem 1.3. *Let* $\mathcal{D} \in \{\mathrm{E}, \mathrm{EXP}, \mathrm{ESPACE}, \mathrm{EXPSPACE}\}$. *For every* $A \in \mathcal{D}$,

$$\dim^{(-3)}(\mathrm{P_m}(A) \mid \mathcal{D}) = 0$$

*or*

$$\dim^{(-3)}(\mathrm{P_m^{-1}}(A) \mid \mathcal{D}) = 0.$$

*In particular,* $\dim^{(-3)}(\deg^{\mathrm{p}}_{\mathrm{m}}(A) \mid \mathcal{D}) = 0$.

In contrast, Theorem 1.2 is extended to scaled dimension at orders $i$ with $|i| \leq 2$.

Theorem 1.4. *Let* $\mathcal{D} \in \{\mathrm{E}, \mathrm{EXP}, \mathrm{ESPACE}, \mathrm{EXPSPACE}\}$. *For every* $A \in \mathcal{D}$ *and* $-2 \leq i \leq 2$,

$$\dim^{(i)}(\deg^{\mathrm{p}}_{\mathrm{m}}(A) \mid \mathcal{D}) = \dim^{(i)}(\mathrm{P_m}(A) \mid \mathcal{D}).$$

This implies that Theorem 1.3 cannot be improved to −2nd-order scaled dimension.

As an application of these results, we consider the scaled dimension of $\mathcal{C}^{\mathrm{p}}_{\mathrm{m}}(\mathrm{E})$, the class of polynomial-time many-one complete sets for E, within ESPACE. Let $i \in \{-2, -1\}$. We extend a theorem of Lutz [15] to show that

$$\dim^{(i)}(\mathcal{C}^{\mathrm{p}}_{\mathrm{m}}(\mathrm{E}) \mid \mathrm{ESPACE}) > 0 \Rightarrow \mathrm{P} = \mathrm{BPP}.$$

On the other hand, we show that

$$\dim^{(i)}(\mathcal{C}^{\mathrm{p}}_{\mathrm{m}}(\mathrm{E}) \mid \mathrm{ESPACE}) < 1 \Rightarrow \mathrm{P} \neq \mathrm{PSPACE}.$$

Therefore, determining the −1st- or −2nd-order scaled dimension of $\mathcal{C}^{\mathrm{p}}_{\mathrm{m}}(\mathrm{E})$ in ESPACE would derandomize BPP or separate P from PSPACE. In contrast, we also show that

$$\dim^{(-3)}(\mathcal{C}^{\mathrm{p}}_{\mathrm{m}}(\mathrm{E}) \mid \mathrm{ESPACE}) = 0$$

and

$$\dim^{(-2)}(\mathcal{C}^{\mathrm{p}}_{\mathrm{m}}(\mathrm{E}) \mid \mathrm{E}) = \dim^{(-1)}(\mathcal{C}^{\mathrm{p}}_{\mathrm{m}}(\mathrm{E}) \mid \mathrm{E}) = 1$$

hold without any hypothesis.

This paper is organized as follows. Section 2 contains the basic preliminaries, and section 3 reviews resource-bounded scaled dimension. We develop some new tools for computing scaled dimension in section 4. The scaled dimensions of some auxiliary classes involving polynomial reductions are calculated in section 5. Our small span theorem for scaled dimension is proved in section 6. Section 7 shows that lower spans and degrees have the same dimension in orders $i$ with $-2 \leq i \leq 2$. Extensions of the results to $\leq^{\mathrm{p}}_{1-\mathrm{tt}}$-reductions are discussed in section 8. The results on the scaled dimension of the complete sets for E are presented in section 9. Section 10 concludes with a brief summary.

## 2. Preliminaries.

The set of all finite binary strings is $\{0,1\}^*$. The empty string is denoted by $\lambda$. We use the standard enumeration of binary strings $s_0 = \lambda, s_1 = 0, s_2 = 1, s_3 = 00, \ldots$. The length of a string $x \in \{0,1\}^*$ is denoted by $|x|$. We use the notation $\{0,1\}^{\leq n} = \{x \in \{0,1\}^* \mid |x| \leq n\}$ and $\{0,1\}^{>n} = \{x \in \{0,1\}^* \mid |x| > n\}$.

All *languages* (decision problems) in this paper are encoded as subsets of $\{0,1\}^*$. For a language $A \subseteq \{0,1\}^*$, we define $A_{\leq n} = \{x \in A \big| |x| \leq n\}$. We routinely identify $A$ with its infinite binary characteristic sequence according to the standard enumeration of binary strings. We write $A \upharpoonright n$ for the $n$-bit prefix of the characteristic sequence of $A$, and $A[n]$ for the $n$th-bit of its characteristic sequence.

Let $\leq^{\mathrm{p}}_r$ be a polynomial-time reducibility. For any $A \subseteq \{0,1\}^*$, let

$$\mathrm{P}_r(A) = \{B \subseteq \{0,1\}^* \mid B \leq^{\mathrm{p}}_r A\}$$

be the $\leq^{\mathrm{p}}_r$*-lower span of* $A$,

$$\mathrm{P}_r^{-1}(A) = \{B \subseteq \{0,1\}^* \mid A \leq^{\mathrm{p}}_r B\}$$

be the $\leq^{\mathrm{p}}_r$*-upper span of* $A$, and

$$\deg^{\mathrm{p}}_r(A) = \mathrm{P}_r(A) \cap \mathrm{P}_r^{-1}(A)$$

be the $\leq^{\mathrm{p}}_r$*-degree of* $A$. For any complexity class $\mathcal{D}$, the class of $\leq^{\mathrm{p}}_r$*-hard languages for* $\mathcal{D}$ is

$$\mathcal{H}^{\mathrm{p}}_r(\mathcal{D}) = \{A \subseteq \{0,1\}^* \mid \mathcal{D} \subseteq \mathrm{P}_r(A)\},$$

and the class of $\leq^{\mathrm{p}}_r$*-complete languages for* $\mathcal{D}$ is

$$\mathcal{C}^{\mathrm{p}}_r(\mathcal{D}) = \mathcal{D} \cap \mathcal{H}^{\mathrm{p}}_r(\mathcal{D}).$$

Let resource $\in$ {time, space} and let $t(n)$ be a resource bound. Let $l \in \mathbb{N}$. A function $f : \mathbb{N}^l \times \{0,1\}^* \to [0,\infty) \cap \mathbb{Q}$ is $t(n)$*-resource exactly computable* if there is a Turing machine that computes $f(k_1, \ldots, k_l, w)$ using at most $t(k_1 + \cdots + k_l + |w|)$ resource for all $(k_1, \ldots, k_l, w) \in \mathbb{N}^l \times \{0,1\}^*$. Let $g : \mathbb{N}^l \times \{0,1\}^* \to [0,\infty)$ be a real-valued function. An *approximation* of $g$ is a function $\hat{g} : \mathbb{N}^{l+1} \times \{0,1\}^* \to [0,\infty)$ such that

$$|g(x) - \hat{g}(r,x)| \leq 2^{-r}$$

for all $x \in \mathbb{N}^l \times \{0,1\}^*$ and $r \in \mathbb{N}$. We say that $g$ is $t(n)$*-resource computable* if there is an exactly $t(n)$-resource computable approximation $\hat{g}$ of $g$. A family of functions $(f_i : \mathbb{N}^l \times \{0,1\}^* \to [0,\infty) \mid i \in \mathbb{N})$ is *uniformly* $t(n)$*-resource (exactly) computable* if the function $f(i,x) = f_i(x)$ is $t(n)$-resource (exactly) computable.

A function $f$ is p-*computable* (respectively, pspace-*computable*) if it is $O(n^k)$-time (respectively, $O(n^k)$-space) computable for some $k \in \mathbb{N}$, and $f$ is $\mathrm{p}_2$-*computable* (respectively, $\mathrm{p}_2$space-*computable*) if it is $O(2^{(\log n)^k})$-time (respectively, $O(2^{(\log n)^k})$-space) computable for some $k \in \mathbb{N}$. Throughout this paper, unless otherwise specified, $\Delta$ denotes any of the *resource bounds* p, $\mathrm{p}_2$, pspace, or $\mathrm{p}_2$space. The concept of an *exactly* $\Delta$-*computable* function is defined analogously.

**3. Scaled dimension.** Hitchcock, Lutz, and Mayordomo [11] introduced resource-bounded scaled dimension. This section briefly reviews the essentials of this theory.

The principle concept is a *scale*, which is a function $g : H \times [0,\infty) \to \mathbb{R}$, where $H = (a,\infty)$ for some $a \in \mathbb{R} \cup \{-\infty\}$. A scale must satisfy certain properties that are given in [11] and will not be discussed here.

The canonical example of a scale is the function $g_0 : \mathbb{R} \times [0,\infty) \to \mathbb{R}$ defined by $g_0(m,s) = sm$. This scale is used in the standard (unscaled) dimension. Other scales of interest are obtained from $g_0$ by rescaling and reflection operations.

*Definition.* Let $g : H \times [0,\infty) \to \mathbb{R}$ be a scale.

1. The *first rescaling* of $g$ is the scale $g^\# : H^\# \times [0,\infty) \longrightarrow \mathbb{R}$ defined by

$$H^\# = \{2^m \mid m \in H\},$$

$$g^\#(m,s) = 2^{g(\log m, s)}.$$

2. The *reflection* of $g$ is the scale $g^R : H \times [0,\infty) \to \mathbb{R}$ defined by

$$g^R(m,s) = \begin{cases} m + g(m,0) - g(m,1-s) & \text{if } 0 \le s \le 1, \\ g(m,s) & \text{if } s \ge 1. \end{cases}$$

A family of scales, one for each integer, is defined as follows.

*Definition.*

1. For each $i \in \mathbb{N}$, define $a_i$ by the recurrence $a_0 = -\infty, a_{i+1} = 2^{a_i}$.
2. For each $i \in \mathbb{Z}$, define the $i$th *scale* $g_i : (a_{|i|},\infty) \times [0,\infty) \to \mathbb{R}$ by the following recursion:
   (a) $g_0(m,s) = sm$.
   (b) For $i \ge 0$, $g_{i+1} = g_i^\#$.
   (c) For $i < 0$, $g_i = g_{-i}^R$.

For clarity, we compute the first few scales. For all $s \in [0,1]$, if $m > a_{|i|}$, then $g_i(m,s)$ is defined by

$$\begin{aligned} g_3(m,s) &= 2^{2^{(\log\log m)^s}}, \\ g_2(m,s) &= 2^{(\log m)^s}, \\ g_1(m,s) &= m^s, \\ g_0(m,s) &= sm, \\ g_{-1}(m,s) &= m + 1 - m^{1-s}, \\ g_{-2}(m,s) &= m + 2 - 2^{(\log m)^{1-s}}, \\ g_{-3}(m,s) &= m + 4 - 2^{2^{(\log\log m)^{1-s}}}. \end{aligned}$$

Scaled dimension is defined using functions called *scaled gales*. The more familiar concepts of *gales* [18] and *martingales* [16] are special cases in the following definition.

*Definition.* Let $i \in \mathbb{Z}$ and let $s \in [0,\infty)$.

1. An *ith-order scaled s-gale* (briefly, an $s^{(i)}$*-gale*) is a function $d : \{0,1\}^{>a_{|i|}} \to [0,\infty)$ such that for all $w \in \{0,1\}^*$ with $|w| > a_{|i|}$,
$$d(w) = 2^{-\Delta g_i(|w|,s)}[d(w0) + d(w1)],$$
where $\Delta g_i : (a_{|i|},\infty) \times [0,\infty) \to \mathbb{R}$ is defined by
$$\Delta g_i(m,s) = g_i(m+1,s) - g_i(m,s).$$
2. An *s-gale* is an $s^{(0)}$-gale, that is, a function $d : \{0,1\}^* \to [0,\infty)$ satisfying
$$d(w) = 2^{-s}[d(w0) + d(w1)]$$
for all $w \in \{0,1\}^*$.
3. A *martingale* is a 1-gale, that is, a function $d : \{0,1\}^* \to [0,\infty)$ satisfying
$$d(w) = \frac{d(w0) + d(w1)}{2}$$
for all $w \in \{0,1\}^*$.

*Success sets* are a crucial concept for resource-bounded measure and also for scaled dimension.

*Definition.* Let $d : \{0,1\}^{>a} \to [0,\infty)$, where $a \in \mathbb{Z}$.
1. We say that $d$ *succeeds* on a language $A \subseteq \{0,1\}^*$ if
$$\limsup_{n\to\infty} d(A \restriction n) = \infty.$$
2. The *success set* of $d$ is
$$S^\infty[d] = \{A \subseteq \{0,1\}^* \mid d \text{ succeeds on } A\}.$$

Resource-bounded measure is defined using success sets of martingales. Here $\Delta$ denotes any of the resource bounds $\{\mathrm{p}, \mathrm{p}_2, \mathrm{pspace}, \mathrm{p}_2\mathrm{space}\}$, and $R(\Delta)$ is the following exponential-time or exponential-space complexity class:
$$\begin{array}{rcccl} R(\mathrm{p}) & = & \mathrm{E} & = & \mathrm{DTIME}(2^{O(n)}), \\ R(\mathrm{p}_2) & = & \mathrm{EXP} & = & \mathrm{DTIME}(2^{n^{O(1)}}), \\ R(\mathrm{pspace}) & = & \mathrm{ESPACE} & = & \mathrm{DSPACE}(2^{O(n)}), \\ R(\mathrm{p}_2\mathrm{space}) & = & \mathrm{EXPSPACE} & = & \mathrm{DSPACE}(2^{n^{O(1)}}). \end{array}$$

*Definition.* Let $\mathcal{C}$ be a class of languages.
1. We say that $\mathcal{C}$ has $\Delta$*-measure 0* and write $\mu_\Delta(\mathcal{C}) = 0$ if there is a $\Delta$-computable martingale $d$ such that $\mathcal{C} \subseteq S^\infty[d]$.
2. We say that $\mathcal{C}$ has *measure 0 in* $R(\Delta)$ and write $\mu(\mathcal{C} \mid R(\Delta)) = 0$ if $\mu_\Delta(\mathcal{C} \cap R(\Delta)) = 0$.

The *measure conservation theorem* of Lutz [16] asserts that $\mu_\Delta(R(\Delta)) \neq 0$, justifying the definition of measure in $R(\Delta)$ above.

Success sets of scaled gales are used to define scaled dimension.

*Definition.* Let $\mathcal{C}$ be a class of languages and $i \in \mathbb{Z}$.
1. The $i$th-order scaled $\Delta$-dimension of $\mathcal{C}$ is
$$\dim_\Delta^{(i)}(\mathcal{C}) = \inf\left\{ s \;\middle|\; \begin{array}{l} \text{there exists a } \Delta\text{-computable} \\ s^{(i)}\text{-gale } d \text{ for which } \mathcal{C} \subseteq S^\infty[d] \end{array} \right\}.$$

2. The $i$th-order scaled dimension of $\mathcal{C}$ within $R(\Delta)$ is

$$\dim^{(i)}(\mathcal{C} \mid R(\Delta)) = \dim_\Delta^{(i)}(\mathcal{C} \cap R(\Delta)).$$

The 0th-order dimension $\dim_\Delta^{(0)}(\cdot)$ is precisely the dimension $\dim_\Delta(\cdot)$ of Lutz [18], and the other orders are interpreted as rescalings of this concept.

The following lemma relates resource-bounded scaled dimension to resource-bounded measure.

LEMMA 3.1 ([11]). *For any class $\mathcal{C}$ of languages and $i \in \mathbb{Z}$,*

$$\dim_\Delta^{(i)}(\mathcal{C}) < 1 \Rightarrow \mu_\Delta(\mathcal{C}) = 0$$

*and*

$$\dim^{(i)}(\mathcal{C} \mid R(\Delta)) < 1 \Rightarrow \mu(\mathcal{C} \mid R(\Delta)) = 0.$$

The following is another key property of scaled dimension.

THEOREM 3.2 (see [11]). *Let $\mathcal{C}$ be a class of languages and $i \in \mathbb{Z}$. If $\dim_\Delta^{(i+1)}(\mathcal{C}) < 1$, then $\dim_\Delta^{(i)}(\mathcal{C}) = 0$.*

This theorem tells us that for every class $\mathcal{C}$, the sequence of dimensions $\dim_\Delta^{(i)}(\mathcal{C})$ for $i \in \mathbb{Z}$ satisfies exactly one of the following three conditions:

(i) $\dim_\Delta^{(i)}(\mathcal{C}) = 0$ for all $i \in \mathbb{Z}$.
(ii) $\dim_\Delta^{(i)}(\mathcal{C}) = 1$ for all $i \in \mathbb{Z}$.
(iii) There exist $i^* \in \mathbb{Z}$ such that $\dim_\Delta^{(i)}(\mathcal{C}) = 0$ for all $i < i^*$ and $\dim_\Delta^{(i)}(\mathcal{C}) = 1$ for all $i > i^*$.

**4. Measures, log-loss, and scaled dimension.** This section provides some tools involving measures and the log-loss concept that are useful for working with the scaled dimensions. It was shown in [10] that *log-loss unpredictability* is equivalent to dimension. Here we characterize scaled dimension using the log-loss of *measures.* A similar approach to classical fractal dimension using measures has been used by Cutler [7] (see also [8]).

*Definition.* A *measure* is a function $\rho : \{0,1\}^* \to [0,\infty)$ satisfying

$$\rho(w) = \rho(w0) + \rho(w1)$$

for all $w \in \{0,1\}^*$.

Measures have the following fundamental relationship with scaled gales. This extends Schnorr's "likelihood ratio" characterization of martingales [23].

OBSERVATION 4.1. *Let $i \in \mathbb{Z}$ and $s \in [0,\infty)$.*

1. *If $\rho : \{0,1\}^* \to [0,\infty)$ is a measure, then the function $d_\rho : \{0,1\}^{>a_{|i|}} \to [0,\infty)$ defined by*

$$d_\rho(w) = 2^{g_i(|w|,s)}\rho(w)$$

*for all $w \in \{0,1\}^{>a_{|i|}}$ is an $s^{(i)}$-gale.*

2. *If $d : \{0,1\}^{>a_{|i|}} \to [0,\infty)$ is an $s^{(i)}$-gale, then the function $\rho_d : \{0,1\}^* \to [0,\infty)$ defined by*

$$\rho_d(w) = 2^{-g_i(|w|,s)}d(w)$$

*for all* $w \in \{0,1\}^{>a_{|i|}}$ *and*

$$\rho_d(w) = \sum_{|v|=a_{|i|}+1-|w|} \rho_d(wv)$$

*for all* $w \in \{0,1\}^{\leq a_{|i|}}$ *is a measure.*

The following lemma relates the scaled dimension of a class to limits involving scales and logarithms of measures.

LEMMA 4.2. *Let* $\mathcal{C}$ *be a class of languages and let* $i \in \mathbb{Z}$.

1. *If* $s > \dim_\Delta^{(i)}(\mathcal{C})$, *then there is a* $\Delta$*-computable measure* $\rho$ *such that*

$$\limsup_{n\to\infty} \left[g_i(n,s) + \log \rho(A \upharpoonright n)\right] = \infty$$

*for all* $A \in \mathcal{C}$.

2. *If* $s < \dim_\Delta^{(i)}(\mathcal{C})$, *then for any* $\Delta$*-computable measure* $\rho$ *there is an* $A_\rho \in \mathcal{C}$ *such that*

$$\lim_{n\to\infty} \left[g_i(n,s) + \log \rho(A_\rho \upharpoonright n)\right] = -\infty.$$

*Proof.* Let $r$ be rational with $s > r > \dim_\Delta^{(i)}(\mathcal{C})$ and let $d$ be a $\Delta$-computable $r^{(i)}$-gale succeeding on $\mathcal{C}$. Then the measure $\rho_d$ from Observation 4.1 is also $\Delta$-computable. Let $A \in \mathcal{C}$. There are infinitely many $n \in \mathbb{N}$ such that $d(A \upharpoonright n) \geq 1$ since $A \in S^\infty[d]$. For such $n$,

$$\begin{aligned} g_i(n,s) + \log \rho_d(A \upharpoonright n) &= g_i(n,s) - g_i(n,r) + \log d(A \upharpoonright n) \\ &\geq g_i(n,s) - g_i(n,r). \end{aligned}$$

Part 1 follows because $r < s$.

For part 2, let $\rho$ be a $\Delta$-computable measure. Let $t$ be rational with $s < t < \dim_\Delta^{(i)}(\mathcal{C})$ and obtain the $t^{(i)}$-gale $d_\rho$ from Observation 4.1. Then $\mathcal{C} \not\subseteq S^\infty[d_\rho]$ because $d_\rho$ is $\Delta$-computable; thus there are an $A_\rho \in \mathcal{C}$ and a constant $c$ such that $d(A \upharpoonright n) \leq c$ for all $n > a_{|i|}$. Then

$$\begin{aligned} g_i(n,s) + \log \rho(A \upharpoonright n) &= g_i(n,s) - g_i(n,t) + \log d_\rho(A \upharpoonright n) \\ &\leq g_i(n,s) - g_i(n,t) + \log c, \end{aligned}$$

and therefore the claim follows because $s < t$. □

Lemma 4.2 asserts that if the $i$th-order scaled dimension of a class $\mathcal{C}$ is less than $s$, then there is a measure $\rho$ such that for every $A \in \mathcal{C}$, there are prefixes $w \sqsubseteq A$ where the *log-loss* quantity

$$-\log \rho(w)$$

is arbitrarily less than $g_i(|w|, s)$.

It is often convenient to replace computable measures by exactly computable measures. The following lemma is proved in the same way as the exact computation lemma for martingales [13].

LEMMA 4.3. *Let* $\rho$ *be a measure that is computable in* $t(n)$ *time (respectively, space), where* $t(n) \geq n$ *is nondecreasing. Then there is a measure* $\tilde{\rho}$ *that is exactly computable in* $n \cdot t(2n+2)$ *time (respectively, space) such that* $\tilde{\rho}(w) \geq \rho(w)$ *for all* $w \in \{0,1\}^*$.

The measures that are exactly computable within a fixed time or space bound are uniformly exactly computable with slightly more time or space.

LEMMA 4.4. *For any nondecreasing time constructible function $t(n) \geq n$ the family of exactly $t(n)$-time computable measures is uniformly exactly computable in $O(n^2 t(n) \log t(n))$ time. The family of exactly $t(n)$-space computable measures is uniformly exactly computable in $O(t(n))$ space.*

*Proof.* There is a uniform enumeration $(M_i \mid i \in \mathbb{N})$ of all $t(n)$-time clocked Turing machines such that for all $i \in \mathbb{N}$, $M_i(w)$ can be computed in $i \cdot t(|w|) \log t(|w|)$ time for all $w \in \{0,1\}^*$. Define $\rho_i : \{0,1\}^* \to [0,\infty)$ inductively by $\rho_i(\lambda) = M_i(\lambda)$ and

$$\rho_i(w0) = \begin{cases} M_i(w0) & \text{if } M_i(w0) \leq \rho_i(w), \\ \rho_i(w) & \text{otherwise,} \end{cases}$$

$$\rho_i(w1) = \rho_i(w) - \rho_i(w0)$$

for all $w \in \{0,1\}^*$. Then each $\rho_i$ is a measure. Also, if $\nu$ is a measure that is exactly computed by $M_i$ in $t(n)$ time, then $\rho_i(w) = \nu(w)$ for all $w$. We can compute $\rho_i(w)$ by using $|w|$ computations of $M_i$ on strings of length at most $|w|$; thus the function $\rho : \mathbb{N} \times \{0,1\}^* \to [0,\infty)$ defined by $\rho(i,w) = \rho_i(w)$ is computable in $O(n^2 t(n) \log t(n))$ time. The argument for space is similar. □

Uniformly exactly computable families of measures can be combined into a single measure in an efficient manner.

LEMMA 4.5. *Let $(\rho_k \mid k \in \mathbb{N})$ be a uniformly exactly $\Delta$-computable family of measures. There is a $\Delta$-computable measure $\rho^*$ such that for any $k$, there is a constant $c_k$ such that*

$$\log \rho^*(w) \geq \log \rho_k(w) - c_k$$

*for all $w \in \{0,1\}^*$.*

*Proof.* Define

$$\rho^*(w) = \sum_{k=0}^{\infty} \frac{\rho_k(w)}{2^k \rho_k(\lambda)}.$$

Then $\rho$ is a measure by linearity. Also, $\rho^*$ is $\Delta$-computable by the approximation function $\hat{\rho}^* : \mathbb{N} \times \{0,1\}^* \to [0,\infty)$ defined by

$$\hat{\rho}^*(r,w) = \sum_{k=0}^{r} \frac{\rho_k(w)}{2^k \rho_k(\lambda)}$$

since

$$\begin{aligned} \left|\rho^*(w) - \hat{\rho}^*(r,w)\right| &= \sum_{k=r+1}^{\infty} \frac{\rho_k(w)}{2^k \rho_k(\lambda)} \\ &\leq \sum_{k=r+1}^{\infty} \frac{\rho_k(\lambda)}{2^k \rho_k(\lambda)} \\ &= 2^{-r}. \end{aligned}$$

Let $k \in \mathbb{N}$. For any $w \in \{0,1\}^*$,

$$\begin{aligned}\log \rho^*(w) &\geq \log \frac{\rho_k(w)}{2^k \rho_k(\lambda)} \\ &= \log \rho_k(w) - k - \rho_k(\lambda),\end{aligned}$$

and thus the lemma holds with $c_k = k + \rho_k(\lambda)$. □

We now combine the preceding lemmas to obtain a tool that will be useful in calculating scaled dimensions.

THEOREM 4.6. *Let $\mathcal{C}$ be a class of languages, $i \in \mathbb{Z}$, and $k \in \mathbb{N}$.*

1. *If for each $A \in \mathcal{C}$ there is a measure $\rho_A$ computable in $O(n^k)$ time such that*

$$(\exists c_A \in \mathbb{Z})(\exists^\infty n) g_i(n,s) + \log \rho_A(A \upharpoonright n) \geq c_A, \tag{4.1}$$

   *then $\dim_{\mathrm{p}}^{(i)}(\mathcal{C}) \leq s$.*
2. *If for each $A \in \mathcal{C}$ there is a measure $\rho_A$ computable in $O(2^{(\log n)^k})$ time such that (4.1) holds, then $\dim_{\mathrm{p}_2}^{(i)}(\mathcal{C}) \leq s$.*
3. *If for each $A \in \mathcal{C}$ there is a measure $\rho_A$ computable in $O(n^k)$ space such that (4.1) holds, then $\dim_{\mathrm{pspace}}^{(i)}(\mathcal{C}) \leq s$.*
4. *If for each $A \in \mathcal{C}$ there is a measure $\rho_A$ computable in $O(2^{(\log n)^k})$ space such that (4.1) holds, then $\dim_{\mathrm{p}_2\mathrm{space}}^{(i)}(\mathcal{C}) \leq s$.*

*Proof.* From Lemmas 4.3, 4.4, and 4.5 we obtain an exactly $\Delta$-computable measure $\rho$ such that $\log \rho(w) \geq \log \rho_A(w) - b_A$ for all $w \in \{0,1\}^*$ where $b_A$ is a constant that depends on $A$ but not on $w$.

Let $t > s$. For any $A \in \mathcal{C}$,

$$g_i(n,t) + \log \rho(A \upharpoonright n) \geq g_i(n,t) - g_i(n,s) + c_A - b_A$$

for infinitely many $n$. Therefore

$$\limsup_{n \to \infty} g_i(n,t) + \log \rho(A \upharpoonright n) = \infty$$

since $t > s$. It follows from the contrapositive of Lemma 4.2(2) that $\dim_\Delta(\mathcal{C}) \leq t$. □

**5. Scaled non-bi-immunity and compressibility.** In this section we introduce some classes involving scales, non-bi-immunity, and compressibility by polynomial-time reductions and calculate their scaled dimensions.

A Turing machine $M$ is *consistent* with a language $A \subseteq \{0,1\}^*$ if for all $x \in \{0,1\}^*$,

$$M(x) \text{ halts} \iff M(x) = A(x).$$

Let $t$ be a time bound. The *fast set of $M$ with respect to $t$* is

$$F_M^t = \{x \in \{0,1\}^* \mid \mathrm{time}_M(x) \leq t(|x|)\}.$$

Recall that $A$ is *not* DTIME($t$)*-bi-immune* if there is a machine $M$ consistent with $A$ such that $F_M^t$ is infinite.

*Definition.* For any time bound $t$, let $X(t)$ be the class of all languages that are not DTIME($t$)-bi-immune.

Let $A \subseteq \{0,1\}^*$ and $f : \{0,1\}^* \to \{0,1\}^*$. We say that $f$ is a *many-one reduction of* $A$ if there is some $B \subseteq \{0,1\}^*$ such that $x \in A \iff f(x) \in B$. The *collision set* of $f$ is

$$C_f = \{s_i | (\exists j < i) f(s_i) = f(s_j)\}.$$

Recall that $A$ is *compressible by* $\leq_{\mathrm{m}}^{\mathrm{DTIME}(t)}$*-reductions* if there exists an $f \in \mathrm{DTIMEF}(t)$ that is a many-one reduction of $A$ and has $C_f$ infinite [12].

*Definition.* For any time bound $t$, let $C(t)$ be the class of all languages that are compressible by $\leq_{\mathrm{m}}^{\mathrm{DTIME}(t)}$-reductions.

The following theorem asserts that almost every language in E is DTIME($2^{cn}$)-bi-immune [21] and incompressible by $\leq_{\mathrm{m}}^{\mathrm{DTIME}(2^{cn})}$-reductions [12].

THEOREM 5.1 (Mayordomo [21], Juedes and Lutz [12]). *For all* $c \in \mathbb{N}$,

$$\mu_{\mathrm{p}}(X(2^{cn})) = \mu_{\mathrm{p}}(C(2^{cn})) = 0$$

*and*

$$\mu_{\mathrm{p}_2}(X(2^{n^c})) = \mu_{\mathrm{p}_2}(C(2^{n^c})) = 0.$$

The next two definitions introduce scaled versions of $X(t)$ and $C(t)$.

*Definition.* For any $i \in \mathbb{Z}$, $\alpha \in [0,1]$, and time bound $t$, let

$$X_\alpha^{(i)}(t) = \left\{ A \subseteq \{0,1\}^* \;\middle|\; \begin{array}{l} (\exists M) M \text{ is consistent with } A \text{ and} \\ (\exists^\infty n) \#(1, F_M^t \restriction n) \geq n - g_i(n,\alpha) \end{array} \right\}.$$

That is, $X_\alpha^{(i)}(t)$ consists of the languages that are not DTIME($t$)-bi-immune in a particular strong way: for infinitely many $n$, all but $g_i(n,\alpha)$ of the first $n$ strings can be decided in less than $t$ time by a consistent Turing machine.

*Definition.* For any $i \in \mathbb{Z}$, $\alpha \in [0,1]$, and time bound $t$, let

$$C_\alpha^{(i)}(t) = \left\{ A \in \{0,1\}^* \;\middle|\; \begin{array}{l} (\exists f \in \mathrm{DTIMEF}(t))\ f \text{ is a many-one reduction of } A \\ \text{and } (\exists^\infty n) \#(1, C_f \restriction n) \geq n - g_i(n,\alpha) \end{array} \right\}.$$

In other words, $C_\alpha^{(i)}(t)$ is the class of languages compressible by $\leq_{\mathrm{m}}^{\mathrm{DTIME}(t)}$-reductions where for infinitely many $n$, all but $g_i(n,\alpha)$ of the first $n$ strings have downward collisions under some reduction.

For $\alpha < 1$, $X_\alpha^{(i)}(2^n) \subseteq X(2^n)$ and $C_\alpha^{(i)}(2^n) \subseteq C(2^n)$, and thus Theorem 5.1 implies that $X_\alpha^{(i)}(2^n)$ and $C_\alpha^{(i)}(2^n)$ have measure 0. We now refine this by calculating their scaled dimensions.

THEOREM 5.2. *For all* $i \in \mathbb{Z}$, $c \geq 1$, *and* $\alpha \in [0,1]$,

$$\dim_{\mathrm{p}}^{(i)}(X_\alpha^{(i)}(2^{cn})) = \dim_{\mathrm{p}}^{(i)}(C_\alpha^{(i)}(2^{cn})) = \alpha$$

*and*

$$\dim_{\mathrm{p}_2}^{(i)}(X_\alpha^{(i)}(2^{n^c})) = \dim_{\mathrm{p}_2}^{(i)}(C_\alpha^{(i)}(2^{n^c})) = \alpha.$$

*Proof.* We focus on the p-dimension portion of the theorem; the argument for $\mathrm{p}_2$-dimension is identical. Let $\alpha \in (0,1)$ and let $s, t > 0$ be arbitrary rationals with $s < \alpha < t$. It suffices to show that

$$s \leq \dim_{\mathrm{p}}^{(i)}(X_\alpha^{(i)}(2^{cn})) \leq \dim_{\mathrm{p}}^{(i)}(C_\alpha^{(i)}(2^{cn})) \leq t.$$

The inequality $\dim_{\mathrm{p}}^{(i)}(X_\alpha^{(i)}(2^{cn})) \leq \dim_{\mathrm{p}}^{(i)}(C_\alpha^{(i)}(2^{cn}))$ holds because of the inclusion $X_\alpha^{(i)}(2^{cn}) \subseteq C_\alpha^{(i)}(2^{cn})$.

For the lower bound, let $\rho$ be any p-computable measure; assume without loss of generality that $\rho(\lambda) \leq 1$. We define a language $A$ inductively by lengths. Let $s < s'' < s' < \alpha$ with $s'$ rational. The first $\lceil g_i(2^n, s')\rceil$ bits of $A_{=n}$ are set by diagonalization to minimize $\rho$. The remaining $2^n - \lceil g_i(2^n, s')\rceil$ bits are identically 0. More formally, if $x$ is the characteristic string of $A_{\leq n-1}$, we choose $v \in \{0,1\}^{\lceil g_i(2^n,s')\rceil}$ so that $\rho(xv)$ is minimized, and we let $A_{=n}$ have characteristic string $v0^{2^n - \lceil g_i(2^n,s')\rceil}$. Then $A$ is in $X_\alpha^{(i)}(2^{cn})$. Let $w \sqsubseteq A$, and let $n$ be such that $2^n - 1 \leq |w| < 2^{n+1} - 1$. Then

$$\log \rho(w) \leq \left(\sum_{j=0}^{n-1} -g_i(2^j, s')\right) - \min\{|w| - (2^n - 1), g_i(2^n, s')\},$$

which is at most $-g_i(|w|, s'')$ if $|w|$ is sufficiently large by Lemma 5.3 below. Then

$$\log \rho(w) + g_i(|w|, s) \leq -g_i(|w|, s'') + g_i(|w|, s),$$

and thus

$$\lim_{n\to\infty} \log \rho(A \upharpoonright n) + g_i(n, s) = -\infty$$

since $s'' > s$. Since $\rho$ is an arbitrary p-computable measure, the contrapositive of Lemma 4.2(1) implies that $\dim_{\mathrm{p}}^{(i)}(X_\alpha^{(i)}(2^{cn})) \geq s$.

Now we prove the upper bound. Let $A \in C_\alpha^{(i)}(2^{cn})$ by a function $f \in \mathrm{DTIMEF}(2^{cn})$. Define a measure $\rho$ inductively by $\rho(\lambda) = 1$ and for all $w \in \{0,1\}^*, b \in \{0,1\}$,

1. if $f(s_i) \neq f(s_{|w|})$ for all $i < |w|$, then
$$\rho(wb) = \frac{\rho(w)}{2};$$
2. otherwise, let $i = \min\{i < |w| \mid f(s_i) = f(s_{|w|})\}$ and define
$$\rho(wb) = \begin{cases} \rho(w) & \text{if } b = w[i], \\ 0 & \text{if } b \neq w[i]. \end{cases}$$

Then for all $w \sqsubseteq A$,

$$\begin{aligned} \log \rho(w) &= -\#(0, C_f \upharpoonright |w|) \\ &= \#(1, C_f \upharpoonright |w|) - |w|. \end{aligned}$$

Whenever $\#(1, C_f \upharpoonright n) \geq n - g_i(n, \alpha)$, we have

$$\begin{aligned} g_i(n,t) + \log \rho(A \upharpoonright n) &= g_i(n,t) + \#(1, C_f \upharpoonright n) - n \\ &\geq g_i(n,t) - g_i(n,\alpha). \end{aligned}$$

This happens infinitely often; thus

$$\limsup_{n\to\infty} g_i(n,t) + \log \rho(A \upharpoonright n) = \infty$$

because $t > \alpha$. Also, $\rho$ is computable in $O(|w| \cdot 2^{c\log|w|}) = O(|w|^{c+1})$ time. Such a $\rho$ can be defined for each $A \in C_\alpha^{(i)}(2^{cn})$, and thus $\dim_{\mathrm{p}}^{(i)}(C_\alpha^{(i)}(2^{cn})) \leq t$ follows by Theorem 4.6. □

LEMMA 5.3. *Let $i \in \mathbb{Z}$, $0 < r < r' < 1$. Then for all sufficiently large $n$ and $k$ with $2^n - 1 \leq k < 2^{n+1} - 1$,*

$$\sum_{j=0}^{n-1} g_i(2^j, r') + \min\{k - (2^n - 1), g_i(2^n, r')\} \geq g_i(k, r).$$

*Proof.* If $i = 0$, then the left-hand side is

$$r'(2^n - 1) + \min\{k - (2^n - 1), r'2^n\} \geq r'k = g_0(k, r') > g_0(k, r).$$

If $i = 1$, then the left-hand side is

$$\sum_{j=0}^{n-1} 2^{jr'} + \min\{k - (2^n - 1), 2^{nr'}\} > 2^{(n-1)r'} > 2^{(n+1)r} > k^r = g_1(k, r)$$

when $n$ is large enough. The argument for $i > 1$ is similar.

If $i = -1$, then the left-hand side is

$$\begin{aligned} 2^n - 1 - \sum_{j=0}^{n-1} 2^{j(1-r')} + n + \min\{k - (2^n - 1), 2^n - 2^{n(1-r')} + 1\} \\ &\geq k - (n+1)2^{n(1-r')} \\ &\geq k - 2^{n(1-r)} + 1 \\ &= g_{-1}(k, r) \end{aligned}$$

if $n$ is sufficiently large. The argument for $i < -1$ is similar. □

**6. Small span theorem.** In this section we establish our small span theorem for scaled dimension. We begin with a simple, but important, lemma about the scales.

LEMMA 6.1. *For all $k \geq 1$ and $s, t \in (0, 1)$, $g_3(2^{n^k}, s) = o(g_2(2^n, t))$.*

*Proof.* We have

$$g_3(2^{n^k}, s) = 2^{2^{\left(\log\log 2^{n^k}\right)^s}} = 2^{2^{(k\log n)^s}}$$

and

$$g_2(2^n, t) = 2^{(\log 2^n)^t} = 2^{n^t} = 2^{2^{t\log n}}.$$

The lemma holds since $(k \log n)^s = o(t \log n)$. □

Juedes and Lutz [12] proved that the upper spans of incompressible languages are small. Specifically, for any language $A \in \mathrm{EXP}$ that is incompressible by $\leq_{\mathrm{m}}^{\mathrm{p}}$-reductions, they showed that $\mu_{\mathrm{p}_2}(\mathrm{P}_{\mathrm{m}}^{-1}(A)) = 0$, and if additionally $A \in \mathrm{E}$, then

$\mu_{\mathrm{p}}(\mathrm{P}_{\mathrm{m}}^{-1}(A)) = 0$. The following theorem is a scaled dimension analogue of this. For any $i \in \mathbb{Z}$, let

$$C_\alpha^{(i)}(\mathrm{poly}) = \bigcup_{c\in\mathbb{N}} C_\alpha^{(i)}(n^c + c).$$

THEOREM 6.2. *Let $\alpha \in (0,1)$.*

1. *Let $\Delta \in \{\mathrm{p}, \mathrm{pspace}\}$. For any $B \in R(\Delta) - C_\alpha^{(1)}(\mathrm{poly})$, $\dim_\Delta^{(-3)}(\mathrm{P}_{\mathrm{m}}^{-1}(B)) = 0$.*
2. *Let $\Delta \in \{\mathrm{p}_2, \mathrm{p}_2\mathrm{space}\}$. For any $B \in R(\Delta) - C_\alpha^{(2)}(\mathrm{poly})$, $\dim_\Delta^{(-3)}(\mathrm{P}_{\mathrm{m}}^{-1}(B)) = 0$.*

*Proof.* We first give the proof for $\Delta = \mathrm{p}$. Let $B \in \mathrm{E} - C_\alpha^{(1)}(\mathrm{poly})$ and let $M$ be a Turing machine that decides $B$ in $O(2^{cn})$ time. Assume $B \leq_{\mathrm{m}}^{\mathrm{p}} C$ via $f$ where $f$ is computable in $n^k$ time almost everywhere. Then for all sufficiently large $n$,

$$f(\{0,1\}^{\leq n}) \subseteq \{0,1\}^{\leq n^k} \tag{6.1}$$

and

$$\left|f(\{0,1\}^{\leq n})\right| \geq g_1(2^{n+1}-1, \alpha) \geq g_1(2^n, \alpha), \tag{6.2}$$

with the latter holding because $B \notin C_\alpha^{(1)}(\mathrm{poly})$.

Let $r \in \mathbb{N}$ such that $\frac{1}{r} < \alpha$. Define $d : \mathbb{N} \to \mathbb{N}$ by $d(n) = \lfloor n/r \rfloor$. For each $n \in \mathbb{N}$ we define a measure $\rho_n : \{0,1\}^* \to [0,1]$ by

$$\rho_n(\lambda) = 2^{-n}$$

and for all $w \in \{0,1\}^*$ and $b \in \{0,1\}$,

1. If $|w| < 2^{d(n)}$ or $\left[(\forall i < 2^{n+1}-1) f(s_i) \neq f(s_{|w|})\right]$, then
$$\rho_n(wb) = \frac{\rho_n(w)}{2}.$$
2. Otherwise, let $i = \min\left\{ i < 2^{n+1}-1 \,\middle|\, f(s_i) = f(s_{|w|}) \right\}$ and define
$$\rho_n(wb) = \begin{cases} \rho_n(w) & \text{if } b = B[i], \\ 0 & \text{if } b \neq B[i]. \end{cases}$$

If $|w| < 2^{d(n)}$, then $\rho_n(w)$ is computable in $O(|w|)$ time. If $|w| \geq 2^{d(n)}$, we can compute $\rho_n(w)$ by using $2^{n+1} - 1 = O(|w|^{n/d(n)}) = O(|w|^r)$ computations of $M$ and $f$ on strings with length at most $n = O(\log|w|)$. Therefore $\rho_n(w)$ is computable in $O(|w|^r(2^{c\log|w|} + (\log|w|)^k)) = O(|w|^{r+c})$ time for all $w \in \{0,1\}^*$.

Let $w_n = C \restriction 2^{n^k+1} - 1$ be the characteristic string of $C_{\leq n^k}$. Then letting

$$m(n) = \left| \left\{ j < |w_n| \,\middle|\, (\forall i < 2^{n+1}-1) f(s_i) \neq f(s_j) \right\} \right|,$$

we have

$$\rho_n(w_n) \geq \rho_n(\lambda) 2^{-2^{d(n)} - m(n)} = 2^{-2^{d(n)} - m(n) - n}.$$

By (6.1) and (6.2), we have

$$m(n) \leq 2^{n^k+1} - 1 - g_1(2^n, \alpha)$$

if $n$ is sufficiently large. In this case,

$$\log \rho_n(w_n) \geq g_1(2^n, \alpha) - 2^{d(n)} - 2^{n^k+1} - n. \tag{6.3}$$

The function $\rho : \{0,1\}^* \to [0, \infty)$ defined by

$$\rho(w) = \sum_{n=0}^{\infty} \rho_n(w)$$

for all $w$ is a measure by linearity. Notice that $\rho(w)$ can be approximated to a precision of $2^{-l}$ in $O(|w|^{r+c}l)$ time by adding the first $l+1$ terms of the sum.

Using (6.3), for all sufficiently large $n$, we have

$$\begin{aligned} g_{-3}(|w_n|, s) + \log \rho_n(w_n) &= 2^{n^k+1} + 4 - g_3(2^{n^k+1} - 1, 1-s) + \log \rho_n(w_n) \\ &\geq g_1(2^n, \alpha) - g_3(2^{n^k+1} - 1, 1-s) - 2^{d(n)} - n. \end{aligned}$$

By Lemma 6.1, $g_3(2^{n^k+1} - 1, 1-s) = o(g_1(2^n, \alpha))$. Also, $2^{d(n)} = 2^{\lfloor n/r \rfloor}$ is little-$o$ of $g_1(2^n, \alpha) = 2^{\alpha n}$ because $\alpha > 1/r$. Using these facts, it follows that

$$\limsup_{n \to \infty} \; g_{-3}(n, s) + \log \rho_n(C \restriction n) = \infty.$$

Appealing to Theorem 4.6, we establish $\dim_{\mathrm{p}}^{(-3)}(\mathrm{P}_{\mathrm{m}}^{-1}(B)) \leq s$. As $s > 0$ is arbitrary, the $\Delta = \mathrm{p}$ part of the theorem holds. The argument is identical for $\Delta = \mathrm{pspace}$.

The proof for $\Delta \in \{\mathrm{p}_2, \mathrm{p}_2\mathrm{space}\}$ is very similar, so we sketch only the differences for $\Delta = \mathrm{p}_2$. Let $B \in \mathrm{EXP} - C_\alpha^{(2)}(2^n)$ and let $M$ be a Turing machine that decides $B$ in $O(2^{n^c})$ time. Assume $B \leq_{\mathrm{m}}^{\mathrm{p}} C$ via $f$. The measures $\rho_n$ and $\rho$ are defined in the same way, except we use a different function $d(n)$. For this, we let $r > 1/\alpha$ and define $d(n) = \lfloor n^\epsilon \rfloor$, where $\epsilon = 1/r$. Then, if $|w| \geq 2^{d(n)}$, as before we can compute $\rho_n(w)$ by using $2^{n+1} - 1$ computations of $M$ and $f$ on strings with length at most $n = O(\log |w|)$. Since $2^n = 2^{(\log 2^{n^\epsilon})^r} = O(2^{(\log |w|)^r})$, we can compute $\rho_n(w)$ in $O(2^{(\log |w|)^r} \cdot 2^{(\log |w|)^c}) = O(2^{(\log |w|)^{\max(r,c)}})$ time. Instead of (6.3), we arrive at $\log \rho_n(w_n) \geq g_2(2^n, \alpha) - 2^{d(n)} - 2^{n^k+1} - n$. The proof is completed in the same way using the fact that $2^{d(n)} = o(g_2(2^n, \alpha))$ because $\epsilon < \alpha$. $\square$

We are now ready to prove our main theorem.

THEOREM 6.3.

1. *Let* $\Delta \in \{\mathrm{p}, \mathrm{pspace}\}$. *For every* $A \in R(\Delta)$,

$$\dim^{(1)}(\mathrm{P}_{\mathrm{m}}(A) \mid R(\Delta)) = 0$$

*or*

$$\dim^{(-3)}(\mathrm{P}_{\mathrm{m}}^{-1}(A) \mid R(\Delta)) = \dim_\Delta^{(-3)}(\mathrm{P}_{\mathrm{m}}^{-1}(A)) = 0.$$

2. *Let* $\Delta \in \{\mathrm{p}_2, \mathrm{p}_2\mathrm{space}\}$. *For every* $A \in R(\Delta)$,

$$\dim^{(2)}(\mathrm{P}_{\mathrm{m}}(A) \mid R(\Delta)) = 0$$

*or*

$$\dim^{(-3)}(\mathrm{P}_{\mathrm{m}}^{-1}(A) \mid R(\Delta)) = \dim_\Delta^{(-3)}(\mathrm{P}_{\mathrm{m}}^{-1}(A)) = 0.$$

*Proof.* Let $\Delta \in \{\mathrm{p}, \mathrm{pspace}\}$ and let $A \in R(\Delta)$. As in the proof of the small span theorem in [12], we consider two cases.

(I) Suppose that

$$\mathrm{P_m}(A) \cap R(\Delta) \subseteq \bigcap_{\alpha \in (0,1)} C_\alpha^{(1)}(2^n).$$

Then $\dim_\Delta^{(1)}(\mathrm{P_m}(A) \cap R(\Delta)) \leq \dim_\mathrm{p}^{(1)}(C_\alpha^{(1)}(2^n)) \leq \alpha$ by Theorem 5.2 for all $\alpha \in (0,1)$, and thus $\dim^{(1)}(\mathrm{P_m}(A) \mid R(\Delta)) = \dim_\Delta^{(1)}(\mathrm{P_m}(A) \cap R(\Delta)) = 0$.

(II) Otherwise, there is an $\alpha \in (0,1)$ such that

$$\mathrm{P_m}(A) \cap R(\Delta) \not\subseteq C_\alpha^{(1)}(2^n).$$

Let $B \in \mathrm{P_m}(A) \cap R(\Delta) - C_\alpha^{(1)}(2^n)$. Then by Theorem 6.2, $\dim_\Delta^{(-3)}(\mathrm{P_m^{-1}}(B)) = 0$. Since $\mathrm{P_m^{-1}}(A) \subseteq \mathrm{P_m^{-1}}(B)$, we have $\dim_\Delta^{(-3)}(\mathrm{P_m^{-1}}(A)) = 0$.

Part 2 is proved in the same way. □

Theorem 6.3 implies that there is a small span theorem for $-3$rd-order scaled dimension, but it is stronger than the following.

Corollary 6.4. *For every $A \in R(\Delta)$,*

$$\dim^{(-3)}(\mathrm{P_m}(A) \mid R(\Delta)) = 0$$

*or*

$$\dim^{(-3)}(\mathrm{P_m^{-1}}(A) \mid R(\Delta)) = \dim_\Delta^{(-3)}(\mathrm{P_m^{-1}}(A)) = 0.$$

*Proof.* This follows immediately from Theorem 6.3 using Theorem 3.2. □

The small span theorem of Juedes and Lutz [12] is also a corollary.

Corollary 6.5 (Juedes and Lutz [12]). *Let $\Delta \in \{\mathrm{p}, \mathrm{p_2}\}$. For every $A \in R(\Delta)$,*

$$\mu(\mathrm{P_m}(A) \mid R(\Delta)) = 0$$

*or*

$$\mu(\mathrm{P_m^{-1}}(A) \mid R(\Delta)) = \mu_\Delta(\mathrm{P_m^{-1}}(A)) = 0.$$

*Proof.* This follows immediately from Theorem 6.3 and Lemma 3.1. □

We also have the following regarding the scaled dimensions of the hard languages for EXP and NP.

Corollary 6.6.

1. $\dim_\mathrm{p}^{(-3)}(\mathcal{H}_\mathrm{m}^\mathrm{p}(\mathrm{EXP})) = \dim_{\mathrm{p_2}}^{(-3)}(\mathcal{H}_\mathrm{m}^\mathrm{p}(\mathrm{EXP})) = 0$.
2. *If* $\dim^{(1)}(\mathrm{NP} \mid \mathrm{E}) > 0$, *then* $\dim_\mathrm{p}^{(-3)}(\mathcal{H}_\mathrm{m}^\mathrm{p}(\mathrm{NP})) = 0$.
3. *If* $\dim^{(2)}(\mathrm{NP} \mid \mathrm{EXP}) > 0$, *then* $\dim_{\mathrm{p_2}}^{(-3)}(\mathcal{H}_\mathrm{m}^\mathrm{p}(\mathrm{NP})) = 0$.

*Proof.* Let $H \in \mathcal{C}_\mathrm{m}^\mathrm{p}(\mathrm{E})$. Then also $H \in \mathcal{C}_\mathrm{m}^\mathrm{p}(\mathrm{EXP})$, so $\mathrm{P_m^{-1}}(H) = \mathcal{H}_\mathrm{m}^\mathrm{p}(\mathrm{EXP})$. Since $\dim(\mathrm{P_m}(H) \mid \mathrm{E}) = \dim_\mathrm{p}(\mathrm{E}) = 1$, Theorem 6.3 tells us that $\dim_\mathrm{p}(\mathcal{H}_\mathrm{m}^\mathrm{p}(\mathrm{EXP})) = \dim_\mathrm{p}(\mathrm{P_m^{-1}}(H)) = 0$.

Parts 2 and 3 follow from Theorem 6.3 using any NP-complete language $A$. □

Juedes and Lutz [12] concluded from their small span theorem that *every* $\leq_\mathrm{m}^\mathrm{p}$-degree has measure 0 in E and in EXP. From Theorem 6.3 we similarly derive a stronger version of this fact: every $\leq_\mathrm{m}^\mathrm{p}$-degree actually has $-3$rd-order dimension 0.

Corollary 6.7. *For every $A \subseteq \{0,1\}^*$,*

$$\dim^{(-3)}(\deg_{\mathrm{m}}^{\mathrm{p}}(A) \mid R(\Delta)) = 0.$$

*Proof.* If $\deg_{\mathrm{m}}^{\mathrm{p}}(A)$ is disjoint from $R(\Delta)$, then $\dim^{(-3)}(\deg_{\mathrm{m}}^{\mathrm{p}}(A) \mid R(\Delta)) = \dim_{\mathrm{p}}^{(-3)}(\emptyset) = 0$, so assume that there is some $B \in \deg_{\mathrm{m}}^{\mathrm{p}}(A) \cap R(\Delta)$. Because $\deg_{\mathrm{m}}^{\mathrm{p}}(A) = \deg_{\mathrm{m}}^{\mathrm{p}}(B) = \mathrm{P}_{\mathrm{m}}(B) \cap \mathrm{P}_{\mathrm{m}}^{-1}(B)$, we have

$$\dim^{(-3)}(\deg_{\mathrm{m}}^{\mathrm{p}}(A) \mid R(\Delta)) \leq \dim^{(-3)}(\mathrm{P}_{\mathrm{m}}(B) \mid R(\Delta))$$

and

$$\dim^{(-3)}(\deg_{\mathrm{m}}^{\mathrm{p}}(A) \mid R(\Delta)) \leq \dim^{(-3)}(\mathrm{P}_{\mathrm{m}}^{-1}(B) \mid R(\Delta)).$$

By Corollary 6.4, we have either $\dim^{(-3)}(\mathrm{P}_{\mathrm{m}}(B) \mid R(\Delta)) = 0$ or $\dim^{(-3)}(\mathrm{P}_{\mathrm{m}}^{-1}(B) \mid R(\Delta)) = 0$. Therefore $\dim^{(-3)}(\deg_{\mathrm{m}}^{\mathrm{p}}(A) \mid R(\Delta)) = 0$. □

The $\leq_{\mathrm{m}}^{\mathrm{p}}$-complete languages for any complexity class have −3rd-order dimension 0 in every $R(\Delta)$.

Corollary 6.8. *For any class $\mathcal{D}$ of languages,* $\dim^{(-3)}(\mathcal{C}_{\mathrm{m}}^{\mathrm{p}}(\mathcal{D}) \mid R(\Delta)) = 0$.

*Proof.* If $\mathcal{C}_{\mathrm{m}}^{\mathrm{p}}(\mathcal{D}) = \emptyset$, this is trivial. Assume $\mathcal{C}_{\mathrm{m}}^{\mathrm{p}}(\mathcal{D}) \neq \emptyset$ and let $A \in \mathcal{C}_{\mathrm{m}}^{\mathrm{p}}(\mathcal{D})$. Then $\mathcal{C}_{\mathrm{m}}^{\mathrm{p}}(\mathcal{D}) \subseteq \deg_{\mathrm{m}}^{\mathrm{p}}(A)$, so this follows from Corollary 6.7. □

**7. Lower spans versus degrees in orders −2 through 2.** We now present some results that stand in contrast to the small span theorem of the previous section. We begin by extending the work of Ambos-Spies et al. [2] to show that lower spans and degrees have the same scaled dimension in orders $i$ with $|i| \leq 2$.

Theorem 7.1. *For any $A \in R(\Delta)$ and $-2 \leq i \leq 2$,*

$$\dim^{(i)}(\deg_{\mathrm{m}}^{\mathrm{p}}(A) \mid R(\Delta)) = \dim^{(i)}(\mathrm{P}_{\mathrm{m}}(A) \mid R(\Delta))$$

*and*

$$\dim_{\Delta}^{(i)}(\deg_{\mathrm{m}}^{\mathrm{p}}(A)) = \dim_{\Delta}^{(i)}(\mathrm{P}_{\mathrm{m}}(A)).$$

*Proof.* We write the proof for dimension in $R(\mathrm{p}) = \mathrm{E}$; the rest of the theorem is proved in the same manner. The proof is based on [2].

Let $A \in \mathrm{E}$ be decidable in $O(2^{cn})$ time. By monotonicity, $\dim^{(i)}(\deg_{\mathrm{m}}^{\mathrm{p}}(A) \mid \mathrm{E}) \leq \dim^{(i)}(\mathrm{P}_{\mathrm{m}}(A) \mid \mathrm{E})$. For the other inequality, let $t > s > \dim^{(i)}(\deg_{\mathrm{m}}^{\mathrm{p}}(A) \mid \mathrm{E})$. By Lemmas 4.2 and 4.3, for some $l \in \mathbb{N}$ there is an exactly $n^l$-time computable measure $\rho$ satisfying

$$\limsup_{m\to\infty} g_i(m,s) + \log \rho(C \upharpoonright m) = \infty \tag{7.1}$$

for all $C \in \deg_{\mathrm{m}}^{\mathrm{p}}(A) \cap \mathrm{E}$.

Letting $k \geq 1$ be a natural number to be specified later, we define a padding function $f : \{0,1\}^* \to \{0,1\}^*$ by

$$f(x) = 0^{|x|^k - |x|}x$$

for all $x$. Let $R = f(\{0,1\}^*)$ be the range of $f$.

Let $B \in \mathrm{P_m}(A)$. We define another language $B'$ as

$$B' = (B - R) \cup f(A).$$

Then $B' \in \deg_{\mathrm{m}}^{\mathrm{p}}(A)$. Intuitively, $B'$ is a language that is very similar to $B$ but has $A$ encoded sparsely in it. Define a function $\tau : \{0,1\}^* \to \{0,1\}^*$ inductively by $\tau(\lambda) = 1$ and

$$\tau(wb) = \begin{cases} \tau(w)b & \text{if } s_{|w|} \notin R, \\ \tau(w)1 & \text{if } s_{|w|} \in R \cap B', \\ \tau(w)0 & \text{if } s_{|w|} \in R - B' \end{cases}$$

for all $w \in \{0,1\}^*$ and $b \in \{0,1\}$. Notice that

$$\tau(B \upharpoonright n) = B' \upharpoonright n$$

for all $n$.

Define a measure $\gamma$ by $\gamma(\lambda) = 1$ and

$$\gamma(wb) = \begin{cases} \frac{\gamma(w)}{2} & \text{if } s_{|w|} \in R, \\ \frac{\rho(\tau(w)b)}{\rho(\tau(w))}\gamma(w) & \text{if } s_{|w|} \notin R \end{cases}$$

for all $w \in \{0,1\}^*$ and $b \in \{0,1\}$. Intuitively, $\gamma$ is designed to have performance on $B$ that is similar to $\rho$'s performance on $B'$. This is done by mimicking the conditional probabilities of $\rho$ for strings that are not in $R$. Note that $\gamma(w)$ can be exactly computed in $O(|w| \cdot (|w|^l + 2^{c \log |w|}) = O(|w|^{\max(l,c)+1})$ time.

Let $n \in N$ and let $2^{(n-1)^k+1} \leq m \leq 2^{n^k+1} - 1$. Then

$$\begin{aligned} \log \gamma(B \upharpoonright m) &= \sum_{1 \leq i \leq m} \log \frac{\gamma(B \upharpoonright i)}{\gamma(B \upharpoonright i-1)} \\ &= \sum_{\substack{1 \leq i \leq m \\ s_i \notin R}} \log \frac{\rho(\tau(B \upharpoonright i-1)B[i])}{\rho(\tau(B \upharpoonright i-1))} + \sum_{\substack{1 \leq i \leq m \\ s_i \in R}} \log \frac{1}{2} \\ &= \sum_{\substack{1 \leq i \leq m \\ s_i \notin R}} \log \frac{\rho(B' \upharpoonright i)}{\rho(B' \upharpoonright i-1)} - \left|\{1 \leq i \leq m \mid s_i \in R\}\right| \\ &\geq \sum_{1 \leq i \leq m} \log \frac{\rho(B' \upharpoonright i)}{\rho(B' \upharpoonright i-1)} - \left|\{1 \leq i \leq 2^{n^k+1} - 1 \mid s_i \in R\}\right| \\ &= \log \rho(B' \upharpoonright m) - \sum_{i=0}^{n} 2^n \\ &= \log \rho(B' \upharpoonright m) - 2^{n+1} + 1. \end{aligned}$$

Now assume that $g_i(m,s) + \log \rho(B' \upharpoonright m) \geq 1$. Then we have $g_i(m,t) + \log \gamma(B \upharpoonright m) \geq 1$ if

$$2^{n+1} + g_i(m,s) < g_i(m,t). \tag{7.2}$$

To establish

$$\limsup_{n\to\infty} g_i(m,t) + \log \gamma(B \upharpoonright m) \geq 1, \tag{7.3}$$

it now suffices to show we can choose $k$ so that (7.2) holds for all sufficiently large $m$. For each $-2 \le i \le 2$, we now give an appropriate choice of $k$ that yields this.

- $i = 2$: Let $k > 1/t$. Then $g_2(m,t) \ge g_2(2^{(n-1)^k}, t) = 2^{(n-1)^{kt}}$, so $2^{n+1} = o(g_2(m,t))$ because $kt > 1$. Also, $g_2(m,s) = o(g_2(m,t))$ since $s < t$, so (7.2) holds when $m$ is sufficiently large.
- $i = 1$: Let $k = 2$. Then $g_1(m,t) \ge g_1(2^{(n-1)^2}, t) = 2^{t(n-1)^2}$, so $2^{n+1} = o(g_1(m,t))$. Also, $g_1(m,s) = o(g_1(m,t))$, so (7.2) holds for sufficiently large $m$.
- $i = 0$: Let $k = 2$. Then $g_0(m,t) \ge g_0(2^{(n-1)^2}, t) = t2^{(n-1)^2}$, so $2^{n+1} = o(g_0(m,t))$. Also, $g_0(m,s) = o(g_0(m,t))$, so (7.2) holds for sufficiently large $m$.
- $i = -1$: We have $g_{-1}(m,t) = m + 1 - g_1(m, 1-t)$, so (7.2) is true if $2^{n+1} + g_1(m, 1-t) < g_1(m, 1-s)$. Taking $k = 2$, this follows from the argument for $i = 1$ above since $1 - s > 1 - t$.
- $i = -2$: Just as in the $i = -1$ case, (7.2) is true if $2^{n+1} + g_2(m, 1-t) < g_2(m, 1-s)$. Taking $k > 1/(1-s)$, this follows from the argument for $i = 2$ above since $1 - s > 1 - t$.

For each $B \in \mathrm{P_m}(A)$, we have given a $O(n^{\max(l,c)})$-time computable measure $\gamma$ such that (7.3) holds. By Theorem 4.6, $\dim^{(i)}(\mathrm{P_m}(A) \mid \mathrm{E}) \le t$. As $t > \dim^{(i)}(\deg^{\mathrm{p}}_{\mathrm{m}}(A) \mid \mathrm{E})$ is arbitrary, this establishes $\dim^{(i)}(\mathrm{P_m}(A) \mid \mathrm{E}) \le \dim^{(i)}(\deg^{\mathrm{p}}_{\mathrm{m}}(A) \mid \mathrm{E})$. $\square$

Theorem 7.1 for (unscaled) dimension was proved in [2] for $\Delta = \mathrm{p}$.

Corollary 7.2 (Ambos-Spies et al. [2]). *For any $A \in \mathrm{E}$,*

$$\dim(\deg^{\mathrm{p}}_{\mathrm{m}}(A) \mid \mathrm{E}) = \dim(\mathrm{P_m}(A) \mid \mathrm{E})$$

*and*

$$\dim_{\mathrm{p}}(\deg^{\mathrm{p}}_{\mathrm{m}}(A)) = \dim_{\mathrm{p}}(\mathrm{P_m}(A)).$$

Theorem 7.1 implies that Theorem 6.3 cannot be improved in one respect. For any $i, j \in \mathbb{Z}$, let $\mathrm{SST}[i,j]$ be the following statement.

$\mathrm{SST}[i,j]$: *For every $A \in \mathrm{E}, \dim^{(i)}(\mathrm{P_m}(A) \mid \mathrm{E}) = 0$ or $\dim^{(j)}(\mathrm{P_m^{-1}}(A) \mid \mathrm{E}) = 0$.*

Let $H \in \mathcal{C}^{\mathrm{p}}_{\mathrm{m}}(\mathrm{E})$. Then

$$\dim^{(i)}(\mathrm{P_m}(H) \mid \mathrm{E}) = \dim^{(i)}(\mathrm{E} \mid \mathrm{E}) = 1$$

for all $i$ and $\dim^{(-2)}(\deg^{\mathrm{p}}_{\mathrm{m}}(H) \mid \mathrm{E}) = 1$ by Theorem 7.1, which in turn implies

$$\dim^{(-2)}(\mathrm{P_m^{-1}}(H) \mid \mathrm{E}) = 1.$$

Therefore, $\mathrm{SST}[i,j]$ is false if $j \ge -2$. Theorem 6.3 says that $\mathrm{SST}[1,-3]$ is true; now we know that the $-3$ in it cannot be improved to $-2$.

We have the following corollary regarding the classes of complete sets for E, EXP, and NP.

Corollary 7.3. *Let $-2 \le i \le 2$.*

1. $\dim^{(i)}(\mathcal{C}^{\mathrm{p}}_{\mathrm{m}}(\mathrm{E}) \mid \mathrm{E}) = \dim^{(i)}(\mathcal{C}^{\mathrm{p}}_{\mathrm{m}}(\mathrm{EXP}) \mid \mathrm{EXP}) = 1$.
2. $\dim^{(i)}(\mathrm{NP} \mid \mathrm{E}) = \dim^{(i)}(\mathcal{C}^{\mathrm{p}}_{\mathrm{m}}(\mathrm{NP}) \mid \mathrm{E})$.
3. $\dim^{(i)}(\mathrm{NP} \mid \mathrm{EXP}) = \dim^{(i)}(\mathcal{C}^{\mathrm{p}}_{\mathrm{m}}(\mathrm{NP}) \mid \mathrm{EXP})$.

*Proof.* Let $H \in \mathcal{C}^{\mathrm{p}}_{\mathrm{m}}(\mathrm{E})$. Then $\mathcal{C}^{\mathrm{p}}_{\mathrm{m}}(\mathrm{E}) = \deg^{\mathrm{p}}_{\mathrm{m}}(H) \cap \mathrm{E}$, so $\dim^{(i)}(\mathcal{C}^{\mathrm{p}}_{\mathrm{m}}(\mathrm{E}) \mid \mathrm{E}) = \dim^{(i)}(\deg^{\mathrm{p}}_{\mathrm{m}}(H) \mid \mathrm{E}) = \dim^{(i)}(\mathrm{P}_{\mathrm{m}}(H) \mid \mathrm{E}) = \dim^{(i)}_{\mathrm{p}}(\mathrm{E}) = 1$ by Theorem 7.1. The other statements follow similarly. □

We can now observe a difference between the −3rd- and −2nd-order scaled dimensions regarding complete degrees. Corollaries 6.8 and 7.3 together with Theorem 3.2 tell us that for $\mathcal{D} \in \{\mathrm{E}, \mathrm{EXP}\}$,

$$\dim^{(i)}(\mathcal{C}^{\mathrm{p}}_{\mathrm{m}}(\mathcal{D}) \mid \mathcal{D}) = \begin{cases} 0 & \text{if } i \leq -3, \\ 1 & \text{if } i \geq -2 \end{cases}$$

and

$$\dim^{(i)}(\mathcal{C}^{\mathrm{p}}_{\mathrm{m}}(\mathrm{NP}) \mid \mathcal{D}) = \begin{cases} 0 & \text{if } i \leq -3, \\ \dim^{(i)}(\mathrm{NP} \mid \mathcal{D}) & \text{if } i \geq -2. \end{cases}$$

In section 9 we will discuss the scaled dimension of $\mathcal{C}^{\mathrm{p}}_{\mathrm{m}}(\mathrm{E})$ within ESPACE. The following extension of Theorem 7.1 will be useful.

THEOREM 7.4. *For all* $-2 \leq i \leq 2$,

$$\dim^{(i)}(\mathcal{C}^{\mathrm{p}}_{\mathrm{m}}(\mathrm{E}) \mid \mathrm{ESPACE}) = \dim^{(i)}(\mathrm{E} \mid \mathrm{ESPACE}).$$

*Proof.* We use the construction from the proof of Theorem 7.1. Let $t > s > \dim^{(i)}(\mathcal{C}^{\mathrm{p}}_{\mathrm{m}}(\mathrm{E}) \mid \mathrm{ESPACE})$ and take an exactly $n^l$-space computable measure $\rho$ satisfying (7.1) for all $C \in \mathcal{C}^{\mathrm{p}}_{\mathrm{m}}(\mathrm{E})$. Fix an $A \in \mathcal{C}^{\mathrm{p}}_{\mathrm{m}}(\mathrm{E})$. For any $B \in \mathrm{E}$, the set $B'$ constructed from $A$ and $B$ is in $\mathcal{C}^{\mathrm{p}}_{\mathrm{m}}(\mathrm{E})$. The arguments then show $\dim^{(i)}(\mathrm{E} \mid \mathrm{ESPACE}) \leq t$. □

**8. $\leq^{\mathrm{p}}_{1-\mathrm{tt}}$-lower spans versus $\leq^{\mathrm{p}}_{\mathrm{m}}$-lower spans.** Theorem 7.1 is also true for most other polynomial-time reducibilities. (This fact was mentioned in [2] for Corollary 7.2 when it was proved.) To replace $\leq^{\mathrm{p}}_{\mathrm{m}}$ by $\leq^{\mathrm{p}}_{r}$ in the theorem, we need only to have $B' \in \deg^{\mathrm{p}}_{r}(A)$ for the set $B'$ that was constructed in the proof from $B \in \mathrm{P}_r(A)$. In particular, Theorem 7.1 is true for the $\leq^{\mathrm{p}}_{1-\mathrm{tt}}$ reducibility. In this section we show that this holds because of another reason: the scaled dimensions of $\leq^{\mathrm{p}}_{1-\mathrm{tt}}$-lower spans and $\leq^{\mathrm{p}}_{\mathrm{m}}$-lower spans are always the same.

The following proposition was used to show that a set is weakly $\leq^{\mathrm{p}}_{\mathrm{m}}$-complete for exponential time if and only if it is $\leq^{\mathrm{p}}_{1-\mathrm{tt}}$-complete.

PROPOSITION 8.1 (Ambos-Spies et al. [1]). *Let* $A \leq^{\mathrm{p}}_{1-\mathrm{tt}} B$. *Then there is a language* $C \in \mathrm{P}$ *such that*

$$\hat{A} = (A \cap C) \cup (A^c \cap C^c) \leq^{\mathrm{p}}_{\mathrm{m}} B.$$

The idea of the following lemma also comes from [1].

LEMMA 8.2. *Let* $i \in \mathbb{Z}$. *Let* $\mathcal{C}, \hat{\mathcal{C}}$ *be classes of languages such that for any* $A \in \mathcal{C}$, *there is some* $C \in R(\Delta)$ *such that* $\hat{A} = (A \cap C) \cup (A^c \cap C^c) \in \hat{\mathcal{C}}$. *Then* $\dim^{(i)}_{\Delta}(\mathcal{C}) \leq \dim^{(i)}_{\Delta}(\hat{\mathcal{C}})$.

*Proof.* We prove this for $\Delta = \mathrm{p}$. The other cases are proved by identical arguments.

Let $s > \dim^{(i)}_{\mathrm{p}}(\hat{\mathcal{C}})$ be rational and obtain $\rho$ computable in $O(n^r)$ time from Lemma 4.2 such that

$$\limsup_{n\to\infty} g_i(n,s) + \log \rho(\hat{A} \upharpoonright n) = \infty \tag{8.1}$$

for all $\hat{A} \in \hat{\mathcal{C}}$.

Let $A \in \mathcal{C}$ and let $C \in \mathrm{DTIME}(n^k)$ such that $\hat{A} = (A \cap C) \cup (A^c \cap C^c) \in \hat{\mathcal{C}}$. Define a function $\tau : \{0,1\}^* \to \{0,1\}^*$ by

$$\tau(w)[j] = \begin{cases} w[j] & \text{if } s_j \in C, \\ 1 - w[j] & \text{if } s_j \notin C \end{cases}$$

for each $0 \leq j < |w|$. Define another measure $\rho'$ by

$$\rho'(w) = \rho(\tau(w)).$$

Then for all $n$,

$$\rho'(A \upharpoonright n) = \rho(\tau(A \upharpoonright n)) = \rho(\hat{A} \upharpoonright n).$$

Therefore

$$\limsup_{n\to\infty} g_i(n, s) + \log \rho'(A \upharpoonright n) = \infty$$

because of (8.1). As $\rho'$ is computable in time $O(|w| \cdot (\log |w|)^k + |w|^r)$, it follows by Theorem 4.6 that $\dim_{\mathrm{p}}^{(i)}(\mathcal{C}) \leq s$. □

We now show that the scaled dimension of a $\leq_{\mathrm{m}}^{\mathrm{p}}$-lower span is always equal to the scaled dimension of the $\leq_{1-\mathrm{tt}}^{\mathrm{p}}$-lower span.

THEOREM 8.3. *Let $B \subseteq \{0,1\}^*$ and let $i \in \mathbb{Z}$. Then*

$$\dim_{\Delta}^{(i)}(\mathrm{P}_{\mathrm{m}}(B)) = \dim_{\Delta}^{(i)}(\mathrm{P}_{1-\mathrm{tt}}(B))$$

*and*

$$\dim^{(i)}(\mathrm{P}_{\mathrm{m}}(B) \mid R(\Delta)) = \dim^{(i)}(\mathrm{P}_{1-\mathrm{tt}}(B) \mid R(\Delta)).$$

*Proof.* By Proposition 8.1, for each $A \in \mathrm{P}_{1-\mathrm{tt}}(B)$ there is a language $C \in \mathrm{P}$ such that $\hat{A} = (A \cap C) \cup (A^c \cap C^c) \in \mathrm{P}_{\mathrm{m}}(B)$. Let $\hat{\mathcal{C}}$ be the set of all such $\hat{A}$ as $A$ ranges over $\mathrm{P}_{1-\mathrm{tt}}(B)$. Then by Lemma 8.2,

$$\dim_{\Delta}^{(i)}(\mathrm{P}_{1-\mathrm{tt}}(B)) \leq \dim_{\Delta}^{(i)}(\hat{\mathcal{C}}).$$

As $\hat{\mathcal{C}} \subseteq \mathrm{P}_{\mathrm{m}}(B) \subseteq \mathrm{P}_{1-\mathrm{tt}}(B)$, we also have

$$\dim_{\Delta}^{(i)}(\hat{\mathcal{C}}) \leq \dim_{\Delta}^{(i)}(\mathrm{P}_{\mathrm{m}}(B)) \leq \dim_{\Delta}^{(i)}(\mathrm{P}_{1-\mathrm{tt}}(B)),$$

so the first equality holds. The proof for dimension in $R(\Delta)$ is analogous. □

We can now give a stronger version of Theorem 7.1.

COROLLARY 8.4. *For any $A \in R(\Delta)$ and $-2 \leq i \leq 2$,*

$$\begin{array}{ccc} \dim^{(i)}(\mathrm{P}_{\mathrm{m}}(A) \mid R(\Delta)) & = & \dim^{(i)}(\deg_{\mathrm{m}}^{\mathrm{p}}(A) \mid R(\Delta)) \\ \shortparallel & & \shortparallel \\ \dim^{(i)}(\mathrm{P}_{1-\mathrm{tt}}(A) \mid R(\Delta)) & = & \dim^{(i)}(\deg_{1-\mathrm{tt}}^{\mathrm{p}}(A) \mid R(\Delta)), \end{array}$$

*and similarly when $\dim^{(i)}(\cdot \mid R(\Delta))$ is replaced by $\dim_{\Delta}^{(i)}(\cdot)$.*

*Proof.* From Theorems 7.1 and 8.3 we have

$$\dim^{(i)}(\mathrm{deg}^{\mathrm{p}}_{\mathrm{m}}(A) \mid R(\Delta)) = \dim^{(i)}(\mathrm{P}_{\mathrm{m}}(A) \mid R(\Delta)) = \dim^{(i)}(\mathrm{P}_{1-\mathrm{tt}}(A) \mid R(\Delta)).$$

By monotonicity, we have

$$\dim^{(i)}(\mathrm{deg}^{\mathrm{p}}_{\mathrm{m}}(A) \mid R(\Delta)) \le \dim^{(i)}(\mathrm{deg}^{\mathrm{p}}_{1-\mathrm{tt}}(A) \mid R(\Delta)) \le \dim^{(i)}(\mathrm{P}_{1-\mathrm{tt}}(A) \mid R(\Delta)),$$

so the corollary follows. The proof for $\dim^{(i)}_{\Delta}(\cdot)$ is analogous. □

Theorem 8.3 also yields a strengthening of Theorem 6.3: the $\mathrm{P}_{\mathrm{m}}(A)$ in it can be replaced by $\mathrm{P}_{1-\mathrm{tt}}(A)$. In fact, it is also possible to replace the $\mathrm{P}^{-1}_{\mathrm{m}}(A)$ in Theorem 6.3 by $\mathrm{P}^{-1}_{1-\mathrm{tt}}(A)$ by extending Theorems 5.2 and 6.2 to deal with $\le^{\mathrm{p}}_{1-\mathrm{tt}}$-reductions. We omit the details.

## 9. The scaled dimension of $\mathcal{C}^{\mathrm{p}}_{\mathrm{m}}(\mathrm{E})$ in ESPACE.

Lutz [17] proved a small span theorem for nonuniform Turing reductions in ESPACE. This implies that $\mathcal{C}^{\mathrm{p}}_{\mathrm{m}}(\mathrm{E})$ has measure 0 in ESPACE. In Corollary 6.8 we saw that $\mathcal{C}^{\mathrm{p}}_{\mathrm{m}}(\mathrm{E})$ actually has −3rd-order scaled dimension 0 in ESPACE. In this section we show that determining the −2nd- or −1st-order scaled dimension of $\mathcal{C}^{\mathrm{p}}_{\mathrm{m}}(\mathrm{E})$ in ESPACE would yield a proof of P = BPP or P ≠ PSPACE.

The P = BPP hypothesis was related to the measure of E in ESPACE by Lutz [15].

THEOREM 9.1 (Lutz [15]). *If* $\mu(\mathrm{E} \mid \mathrm{ESPACE}) \neq 0$, *then* P = BPP.

We will extend this result to scaled dimension. We now recall the tools Lutz used to prove it.

Nisan and Wigderson [22] showed that BPP can be derandomized if there is a decision problem in E that requires exponential-size circuits to be approximately solved. The *hardness* of a decision problem at a given length is the minimum size of a circuit that can approximately solve it. The details of the definition of this hardness are not needed in this paper; we need only to recall existing results regarding classes of languages with exponential hardness.

*Definition.* Let $H_\alpha$ be the class of all languages that have hardness at least $2^{\alpha n}$ almost everywhere in the sense of [22].

The aforementioned derandomization of BPP can be stated as follows.

THEOREM 9.2 (see Nisan and Wigderson [22]). *If* $\mathrm{E} \cap H_\alpha \neq \emptyset$ *for some* $\alpha > 0$, *then* P = BPP.

We will also need space-bounded Kolmogorov complexity.

*Definition.* Given a machine $M$, a space bound $s : \mathbb{N} \to \mathbb{N}$, a language $L \subseteq \{0,1\}^*$, and a natural number $n$, the *s-space-bounded Kolmogorov complexity* of $L_{=n}$ with respect to $M$ is

$$\mathrm{KS}^s_M(L_{=n}) = \min \left\{ |\pi| \middle| M(\pi, n) = \chi_{L_{=n}} \text{ in } \le s(2^n) \text{ space} \right\},$$

i.e., the length of the shortest program $\pi$ such that $M$, on input $(\pi, n)$, outputs the characteristic string of $L_{=n}$ and halts without using more than $s(2^n)$ workspace.

Well-known simulation techniques show that there exists a machine $U$ which is *optimal* in the sense that for each machine $M$ there is a constant $c$ such that for all $s$, $L$, and $n$ we have

$$\mathrm{KS}^{cs+c}_U(L_{=n}) \le \mathrm{KS}^s_M(L_{=n}) + c.$$

As usual, we fix such a universal machine and omit it from the notation.

*Definition.* For each space bound $s : \mathbb{N} \to \mathbb{N}$ and function $f : \mathbb{N} \to \mathbb{N}$ define the complexity class

$$\mathrm{KS}^{s}_{\mathrm{i.o.}}(f) = \{L \subseteq \{0,1\}^* \mid (\exists^\infty n)\mathrm{KS}^s(L_{=n}) < f(n)\}.$$

Lutz showed that $H_\alpha$ has measure 1 in ESPACE (i.e., that $H_\alpha^c$ has measure 0 in ESPACE) if $\alpha < 1/3$ by showing that languages not in $H_\alpha$ have low space-bounded Kolmogorov complexity.

LEMMA 9.3 (Lutz [15]). *There exist a polynomial $q$ and a constant $c$ such that for all $0 < \alpha < \beta < 1$,*

$$H_\alpha^c \subseteq \mathrm{KS}^{q}_{\mathrm{i.o.}}(2^n - c2^{(1-2\alpha)n} + 2^{\beta n}).$$

The class on the right-hand side in Lemma 9.3 has measure 0 in ESPACE [16]. The scaled dimensions of similar space-bounded Kolmogorov complexity classes were studied in [11].

THEOREM 9.4 (see Hitchcock, Lutz, and Mayordomo [11]). *For any $i \leq -1$, polynomial $q(n) = \Omega(n^2)$, and $\alpha \in [0,1]$,*

$$\dim^{(i)}(\mathrm{KS}^{q}_{\mathrm{i.o.}}(g_i(2^n, \alpha)) \mid \mathrm{ESPACE}) = \alpha.$$

Lemma 9.3 and Theorem 9.4 provide an easy upper bound on the $-1$st-order scaled dimension of $H_\alpha^c$ in ESPACE.

COROLLARY 9.5. *If $0 < \alpha < 1/3$, then*

$$\dim^{(-1)}(H_\alpha^c \mid \mathrm{ESPACE}) \leq 2\alpha.$$

*Proof.* Let $\epsilon > 0$ and $\beta \in (\alpha, 1 - 2\alpha)$. Then for all sufficiently large $n$,

$$\begin{aligned} 2^n - c2^{(1-2\alpha)n} + 2^{\beta n} &< 2^n + 1 - 2^{(1-2\alpha-\epsilon)n} \\ &= g_{-1}(2^n, 2\alpha + \epsilon), \end{aligned}$$

so Lemma 9.3 implies $H_\alpha^c \subseteq \mathrm{KS}^{q}_{\mathrm{i.o.}}(g_{-1}(2^n, 2\alpha+\epsilon))$. Therefore $\dim^{(-1)}(H_\alpha^c \mid \mathrm{ESPACE}) \leq 2\alpha + \epsilon$ by Theorem 9.4. $\square$

We can now state a stronger version of Theorem 9.1. The hypothesis has been weakened, but the conclusion remains the same.

THEOREM 9.6. *If $\dim^{(-1)}(\mathrm{E} \mid \mathrm{ESPACE}) > 0$, then $\mathrm{P} = \mathrm{BPP}$.*

*Proof.* Assume the hypothesis and let $s = \min\{1/2, \dim^{(-1)}(\mathrm{E} \mid \mathrm{ESPACE})\}$. Then by Corollary 9.5, $\mathrm{E} \not\subseteq H_{s/2}^c$, i.e., $\mathrm{E} \cap H_{s/2} \neq \emptyset$. Therefore $\mathrm{P} = \mathrm{BPP}$ by Theorem 9.2. $\square$

We now relate the scaled dimension of $\mathcal{C}^{\mathrm{p}}_{\mathrm{m}}(\mathrm{E})$ to the $\mathrm{P} \overset{?}{=} \mathrm{PSPACE}$ and $\mathrm{P} \overset{?}{=} \mathrm{BPP}$ problems.

THEOREM 9.7. *For $i \in \{-2, -1\}$,*

$$\dim^{(i)}(\mathcal{C}^{\mathrm{p}}_{\mathrm{m}}(\mathrm{E}) \mid \mathrm{ESPACE}) < 1 \Rightarrow \mathrm{P} \neq \mathrm{PSPACE}$$

*and*

$$\dim^{(i)}(\mathcal{C}^{\mathrm{p}}_{\mathrm{m}}(\mathrm{E}) \mid \mathrm{ESPACE}) > 0 \Rightarrow \mathrm{P} = \mathrm{BPP}.$$

*Proof.* From Theorem 7.4 we know that $\dim^{(i)}(\mathcal{C}^{\mathrm{p}}_{\mathrm{m}}(\mathrm{E}) \mid \mathrm{ESPACE}) = \dim^{(i)}(\mathrm{E} \mid \mathrm{ESPACE})$. Also, $\dim^{(i)}(\mathrm{E} \mid \mathrm{ESPACE}) < 1$ implies $\mathrm{E} \neq \mathrm{ESPACE}$, which implies $\mathrm{P} \neq \mathrm{PSPACE}$ [5]. This proves the first implication. The second one follows from Theorem 9.6 since $\dim^{(i)}(\mathcal{C}^{\mathrm{p}}_{\mathrm{m}}(\mathrm{E}) \mid \mathrm{ESPACE}) > 0$ implies $\dim^{(-1)}(\mathrm{E} \mid \mathrm{ESPACE}) > 0$. ▯

In other words, establishing any nontrivial upper or lower bound on $\dim^{(-1)}(\mathcal{C}^{\mathrm{p}}_{\mathrm{m}}(\mathrm{E}) \mid \mathrm{ESPACE})$ or $\dim^{(-2)}(\mathcal{C}^{\mathrm{p}}_{\mathrm{m}}(\mathrm{E}) \mid \mathrm{ESPACE})$ would derandomize BPP or separate P from PSPACE. This is in contrast to the unconditional facts from Corollaries 6.7 and 7.3 that

$$\dim^{(-3)}(\mathcal{C}^{\mathrm{p}}_{\mathrm{m}}(\mathrm{E}) \mid \mathrm{E}) = 0$$

and

$$\dim^{(-2)}(\mathcal{C}^{\mathrm{p}}_{\mathrm{m}}(\mathrm{E}) \mid \mathrm{E}) = \dim^{(-1)}(\mathcal{C}^{\mathrm{p}}_{\mathrm{m}}(\mathrm{E}) \mid \mathrm{E}) = 1.$$

## 10. Conclusion.

Our main results, Theorems 6.3 and 7.1, use resource-bounded scaled dimension to strengthen from both ends the contrasting theorems of Juedes and Lutz [12] and Ambos-Spies et al. [2] regarding spans under polynomial-time reductions.

1. The *small span theorem* for $\leq^{\mathrm{p}}_{\mathrm{m}}$-reductions [12] was strengthened from measure to $-3$rd-order scaled dimension. (In fact, Theorem 6.3 is even stronger than this.)
2. The result that lower spans and degrees have the same dimension [2] was extended to all orders $-2 \leq i \leq 2$ of scaled dimension. This implies that there is no small span theorem in $-2$nd-order scaled dimension.

These results suggest that the contrast between the $-2$nd- and $-3$rd-orders of resource-bounded scaled dimension will be useful for studying complexity classes involving polynomial-time reductions. For example, regarding the many-one complete degree of NP, Corollaries 6.7 and 7.3 say that

$$\dim^{(-3)}(\mathcal{C}^{\mathrm{p}}_{\mathrm{m}}(\mathrm{NP}) \mid \mathrm{E}) = 0$$

and

$$\dim^{(-2)}(\mathcal{C}^{\mathrm{p}}_{\mathrm{m}}(\mathrm{NP}) \mid \mathrm{E}) = \dim^{(-2)}(\mathrm{NP} \mid \mathrm{E}).$$

Scaled dimension therefore provides two different types of dimension for studying NP. The NP-complete degree provides all the dimension of NP in order $-2$, but in order $-3$ the NP-complete degree unconditionally has dimension 0.

**Acknowledgments.** I thank Jack Lutz for suggesting this line of research and Xiaoyang Gu for comments on a draft. I also thank the anonymous referees for very helpful comments and corrections.

## REFERENCES

[1] K. Ambos-Spies, E. Mayordomo, and X. Zheng, *A comparison of weak completeness notions*, in Proceedings of the 11th IEEE Conference on Computational Complexity, Philadelphia, PA, IEEE Computer Society, Los Alamitos, CA, 1996, pp. 171–178.

[2] K. Ambos-Spies, W. Merkle, J. Reimann, and F. Stephan, *Hausdorff dimension in exponential time*, in Proceedings of the 16th IEEE Conference on Computational Complexity, Chicago, IL, IEEE Computer Society, Los Alamitos, CA, 2001, pp. 210–217.

[3] K. Ambos-Spies, H.-C. Neis, and S. A. Terwijn, *Genericity and measure for exponential time*, Theoret. Comput. Sci., 168 (1996), pp. 3–19.

[4] C. H. Bennett and J. Gill, *Relative to a random oracle A,* $\mathrm{P}^A \neq \mathrm{NP}^A \neq$ *co-NP*$^A$ *with probability* 1, SIAM J. Comput., 10 (1981), pp. 96–113.

[5] R. V. Book, *Comparing complexity classes*, J. Comput. System Sci., 9 (1974), pp. 213–229.

[6] H. Buhrman and D. van Melkebeek, *Hard sets are hard to find*, J. Comput. System Sci., 59 (1999), pp. 327–345.

[7] C. D. Cutler, *Strong and weak duality principles for fractal dimension in Euclidean space*, Math. Proc. Cambridge Philos. Soc., 118 (1995), pp. 393–410.

[8] K. Falconer, *Techniques in Fractal Geometry*, John Wiley and Sons, Chichester, UK, 1997.

[9] F. Hausdorff, *Dimension und äußeres Maß*, Math. Ann., 79 (1919), pp. 157–179.

[10] J. M. Hitchcock, *Fractal dimension and logarithmic loss unpredictability*, Theoret. Comput. Sci., 304 (2003), pp. 431–441.

[11] J. M. Hitchcock, J. H. Lutz, and E. Mayordomo, *Scaled dimension and nonuniform complexity*, J. Comput. System Sci., 69 (2004), pp. 97–122.

[12] D. W. Juedes and J. H. Lutz, *The complexity and distribution of hard problems*, SIAM J. Comput., 24 (1995), pp. 279–295.

[13] D. W. Juedes and J. H. Lutz, *Weak completeness in* E *and* $\mathrm{E}_2$, Theoret. Comput. Sci., 143 (1995), pp. 149–158.

[14] W. Lindner, *On the Polynomial Time Bounded Measure of One-Truth-Table Degrees and P-Selectivity*, Diplomarbeit, Technische Universität Berlin, Berlin, Germany, 1993.

[15] J. H. Lutz, *An upward measure separation theorem*, Theoret. Comput. Sci., 81 (1991), pp. 127–135.

[16] J. H. Lutz, *Almost everywhere high nonuniform complexity*, J. Comput. System Sci., 44 (1992), pp. 220–258.

[17] J. H. Lutz, *A small span theorem for P/Poly-Turing reductions*, in Proceedings of the 10th Annual Structure in Complexity Theory Conference, Minneapolis, MN, IEEE Computer Society, Los Alamitos, CA, 1995, pp. 324–330.

[18] J. H. Lutz, *Dimension in complexity classes*, SIAM J. Comput., 32 (2003), pp. 1236–1259.

[19] J. H. Lutz and E. Mayordomo, *Measure, stochasticity, and the density of hard languages*, SIAM J. Comput., 23 (1994), pp. 762–779.

[20] J. H. Lutz and E. Mayordomo, *Cook versus Karp-Levin: Separating completeness notions if NP is not small*, Theoret. Comput. Sci., 164 (1996), pp. 141–163.

[21] E. Mayordomo, *Almost every set in exponential time is P-bi-immune*, Theoret. Comput. Sci., 136 (1994), pp. 487–506.

[22] N. Nisan and A. Wigderson, *Hardness vs. randomness*, J. Comput. System Sci., 49 (1994), pp. 149–167.

[23] C. P. Schnorr, *A survey of the theory of random sequences*, in Basic Problems in Methodology and Linguistics, Univ. Western Ontario Ser. Philos. Sci. 11, R. E. Butts and J. Hintikka, eds., Reidel, Dordrecht, The Netherlands, 1977, pp. 193–211.

SIAM J. COMPUT. © 2004 Society for Industrial and Applied Mathematics
Vol. 34, No. 1, pp. 195–226

# RANDOM WALKS ON TRUNCATED CUBES AND SAMPLING 0-1 KNAPSACK SOLUTIONS[*]

BEN MORRIS[†] AND ALISTAIR SINCLAIR[‡]

**Abstract.** We solve an open problem concerning the mixing time of symmetric random walk on the $n$-dimensional cube truncated by a hyperplane, showing that it is polynomial in $n$. As a consequence, we obtain a fully polynomial randomized approximation scheme for counting the feasible solutions of a 0-1 knapsack problem. The results extend to the case of any fixed number of hyperplanes. The key ingredient in our analysis is a combinatorial construction we call a "balanced almost uniform permutation," which is of independent interest.

**Key words.** random walks, Markov chains, hypercubes, random sampling, knapsack problem, random permutations, balanced permutations

**AMS subject classifications.** 68Q25, 68W25, 68R05, 60J10

**DOI.** 10.1137/S0097539702411915

## 1. Introduction.

For a positive real vector $\mathbf{a} = (a_i)_{i=1}^n$ and real number $b$, let $\Omega$ denote the set of 0-1 vectors $\mathbf{x} = (x_i)_{i=1}^n$ for which

$$\mathbf{a} \cdot \mathbf{x} \equiv \sum_{i=1}^{n} a_i x_i \le b.$$

Geometrically, we can view $\Omega$ as the set of vertices of the $n$-dimensional cube $\{0,1\}^n$ which lie on one side of the hyperplane $\mathbf{a} \cdot \mathbf{x} = b$. Combinatorially, $\Omega$ is the set of feasible solutions to the 0-1 knapsack problem defined by $\mathbf{a}$ and $b$: if we think of the $a_i$ as the weights of a set of $n$ items and $b$ as the capacity (weight limit) of a knapsack, then there is a 1-1 correspondence between vectors $\mathbf{x} \in \Omega$ and subsets of items $X$ whose aggregated weight does not exceed the knapsack capacity, given by $X = \{i : x_i = 1\}$. We shall write $a(X)$ for the weight of $X$, i.e., $a(X) = \sum_{i \in X} a_i$.

This paper is concerned with the natural nearest neighbor random walk on the "truncated cube" $G_\Omega$ (i.e., the subgraph of the cube induced by $\Omega$), in which the probability of moving to any neighbor is $\frac{1}{n}$. In the knapsack terminology, this walk is described by the following transition rule from any state $X \in \Omega$:

1. with probability $\frac{1}{2}$ do nothing; else
2. pick an item $i \in \{1, \ldots, n\}$ uniformly at random (u.a.r.);
3. if $i \in X$, move to $X - \{i\}$; if $i \notin X$ and $a(X \cup \{i\}) \le b$, move to $X \cup \{i\}$; else do nothing.

(We have added a uniform holding probability of $\frac{1}{2}$ at every state to avoid technical issues involving periodicity.)

[*]Received by the editors July 22, 2002; accepted for publication (in revised form) October 20, 2003; published electronically October 28, 2004. A preliminary version of this paper appeared in *Proceedings of the 40th IEEE Symposium on Foundations of Computer Science*, 1999, pp. 230–240.
http://www.siam.org/journals/sicomp/34-1/41191.html

[†]Department of Mathematics, Indiana University, Rawles Hall, Bloomington, IN 47405 (benjmorr @indiana.edu). This work was done while the author was in the Department of Statistics, University of California at Berkeley, and was supported by an NSF graduate fellowship and by NSF grant ECS-9873086.

[‡]Computer Science Division, Soda Hall, University of California at Berkeley, Berkeley, CA 94720-1776 (sinclair@cs.berkeley.edu). The research of this author was supported in part by NSF grants CCR-9505448 and CCR-9820951.

It is easy to check that this walk converges to the uniform distribution over $\Omega$ for any choice of $\mathbf{a}$ and $b$. Our main concern in this paper is with the number of steps required until the distribution is close to uniform, starting from an arbitrary initial state. We refer to this as the *mixing time* of the walk (see section 2 for a precise definition).

The question of determining good bounds on the mixing time of this random walk has been posed as an open problem in several places (e.g., [5, 12, 14, 17]) and is of interest for two main reasons. First, despite the development over the past decade of several powerful tools for analyzing the mixing time of combinatorial Markov chains, which have led to a string of surprising results in the field (see, e.g., [12, 9, 14, 17] for surveys), this natural and deceptively simple example remains a challenge to all existing techniques. In particular, the mixing time is not known to be bounded by any polynomial function of $n$. There is strong geometric intuition that it should be: random walk on the entire cube $\{0,1\}^n$ has a mixing time of only $\mathrm{O}(n \log n)$, and truncation by a hyperplane presumably cannot create "bottlenecks" that would severely slow down convergence. Nonetheless, the best known bound on the mixing time remains $\exp\bigl(\mathrm{O}(\sqrt{n}(\log n)^{5/2})\bigr)$ [5], which beats the trivial bound of $\exp(\mathrm{O}(n))$ but is still exponential.

The second reason for the interest in this random walk is its connection to approximate counting. Because the walk converges to the uniform distribution over $\Omega$, it provides an algorithm for sampling (almost) uniformly at random from $\Omega$: just simulate the walk for sufficiently many steps, starting from an arbitrary vertex of $G_\Omega$, and output the final vertex. (This is often known as the "Markov chain Monte Carlo" paradigm.) The running time of the algorithm will be essentially the mixing time of the walk. Now by a well-known relationship based on self-reducibility, for most natural combinatorial structures (including 0-1 knapsack solutions) the problems of approximate counting and of sampling from an (almost) uniform distribution are polynomial time reducible to one another [13, 12]. Therefore, a proof that the above random walk has mixing time polynomial in $n$ would immediately imply the existence of a polynomial time approximation algorithm (actually a *full polynomial randomized approximation scheme*, or *fpras* [12]) for computing $|\Omega|$, the number of feasible knapsack solutions. This problem is #P-complete in exact form, so an fpras is essentially the best we can hope for. Again, following a string of approximate counting algorithms for #P-complete problems in recent years (many of them based on the Markov chain Monte Carlo technique), this problem remains one of the canonical unresolved examples.

In this paper we prove that the above random walk is indeed rapidly mixing, with a mixing time of $\mathrm{O}(n^{9/2+\epsilon})$ steps for any $\epsilon > 0$. This immediately implies the existence of the first fpras for counting 0-1 knapsack solutions.[1] Along the way we develop some new machinery for bounding the mixing time of Markov chains which we believe will be useful for tackling other examples of a similar flavor, and possibly beyond.

We also present a nontrivial extension of these results to the case of multiple hyperplanes (more precisely, multiple constraints of the form $\mathbf{a}_j \cdot \mathbf{x} \leq b_j$ for nonnegative vectors $\mathbf{a}_j$).[2] Here we are also able to prove a mixing time of $\mathrm{O}(n^c)$ (where $c$ is a

[1]Very recently, and after the appearance of the conference version of this paper [16], Martin Dyer gave an algorithm for random sampling and approximate counting of knapsack solutions with running time only $\mathrm{O}(n^{2.5})$ [4]. However, Dyer's algorithm is based on dynamic programming and gives no insight into the mixing time of the random walk.

constant) for any fixed number of hyperplanes. The exponent $c$ depends on the number $d$ of hyperplanes, but this is inevitable as it is not hard to prove a lower bound of $n^{\Omega(d)}$ on the mixing time. Moreover, it is possible to encode NP-hard problems if the number of hyperplanes is permitted to depend on $n$, so we would not expect any polynomial time sampling algorithm for this case.

To prove rapid mixing we use a technique based on multicommodity flow (see [18]): if we can route unit flow between each pair of vertices $X, Y$ in $G_\Omega$ simultaneously in such a way that no edge carries too much flow, then the random walk is rapidly mixing. This technique is well known, but most previous applications (e.g., [10, 11]) have made use of "degenerate" flows in which all $X \to Y$ flow is routed along a single canonical path (though see [2, 18] for exceptions). Our analysis seems to rely essentially on spreading out the flow along multiple paths.

The key ingredient in our analysis is the specification of these paths, which we achieve via an auxiliary combinatorial construction that we believe is of independent interest and will find further applications elsewhere.[3] Note that a shortest path between a pair of vertices $X, Y$ of $G_\Omega$ can be viewed as a *permutation* of the symmetric difference $X \oplus Y$, the set of items that must be added to or removed from the knapsack in passing from $X$ to $Y$. A natural approach to defining a good flow is to use a *random* permutation, so that the flow is spread evenly among all shortest paths and no edge is overloaded. However, a fundamental problem with this approach is that a random permutation will tend to violate the knapsack constraint, as too many items will have been added at some intermediate point. Slightly less obviously, a symmetric problem arises because a random permutation will tend to remove too many items at some intermediate point, causing congestion among edges of the hypercube near the origin. To avoid these problems, we want our permutations to remain "balanced" in the sense that items are added and removed at approximately the correct rates throughout the path; but we also want them to be "sufficiently random" to ensure a well-spread flow. More specifically, it turns out that we require the distribution of the initial segment $\{\pi(1), \dots, \pi(k)\}$, viewed as an unordered set, to be "almost uniform." We call permutations with these properties *balanced almost uniform* permutations. A main contribution of this paper is to show the existence of such permutations.

The remainder of the paper is structured as follows. We begin with some necessary background on flows and rapid mixing in section 2. Then in section 3 we define the notion of balanced almost uniform permutations and show how to construct them; this section is independent of the random walk analysis and should be of wider interest. We go on to use these balanced permutations to define a good flow for the knapsack random walk in section 4. The extension to multiple constraints is handled in section 5; this involves extending our construction of balanced almost uniform permutations in a nontrivial way from scalar weights to vectors in arbitrary dimension, which is again of independent interest. We conclude with the proofs of some technical lemmas in section 6.

## 2. The mixing time and multicommodity flow.

As indicated earlier, we will view elements of $\Omega$ either as 0-1 vectors $\mathbf{x} = (x_i)_{i=1}^n$ or, more commonly, as subsets $X \subseteq \{1, \dots, n\}$, under the equivalence $X = \{i : x_i = 1\}$. Recall that $a(X) = \sum_{i \in X} a_i$

[2]We mention in passing that all our results extend from the 0-1 case to more general cubes of the form $[0, \dots, L]^n$. This extension is purely technical and does not require any substantial new ideas, so we omit the details.

[3]Indeed, these ideas have already been used by Cryan et al. in the analysis of Markov chain Monte Carlo algorithms for contingency tables [1].

is the weight of $X$ so that $\Omega = \{X : a(X) \leq b\}$. Without loss of generality, we will assume that $a_i \leq b$ for all $i$.

We consider the symmetric random walk defined in the introduction on the portion $G_\Omega$ of the hypercube $\{0,1\}^n$. This walk is connected (all states communicate via the zero vector) and aperiodic (because of the holding probabilities), and since the transition probabilities are symmetric, the distribution at time $t$ converges to the uniform distribution over $\Omega$ as $t \to \infty$, regardless of the initial state. Our goal is to bound the rate of convergence as measured by the *mixing time*, defined as

$$\tau_{\mathrm{mix}} = \max_{X_0} \min \left\{ t : \|P_t - \mathcal{U}\| \leq \tfrac{1}{4} \right\},$$

where $X_0$ is the initial state, $P_t$ is the distribution of the walk at time $t$, $\mathcal{U}$ is the uniform distribution over $\Omega$, and $\|\cdot\|$ denotes variation distance.[4] Thus $\tau_{\mathrm{mix}}$ is the number of steps required, starting from any initial state, to get the variation distance from the uniform distribution down to $\frac{1}{4}$. By standard facts about geometric convergence, $\mathrm{O}(\tau_{\mathrm{mix}} \log \epsilon^{-1})$ steps suffice to reduce the variation distance to any desired $\epsilon$.

Fairly standard techniques (see [18]) allow us to estimate $\tau_{\mathrm{mix}}$ by setting up a suitable multicommodity flow on the underlying graph $G_\Omega$. Our task is to route one unit of flow from $X$ to $Y$ for each ordered pair of vertices $X, Y \in \Omega$ simultaneously. For any such flow $f$ and any oriented edge $e$ in $G_\Omega$, let $f(e)$ denote the total flow along $e$; i.e., $f(e)$ is the sum over all ordered pairs $X, Y$ of the $X \to Y$ flow carried by $e$. Define the *congestion* $\mathcal{C}(f) = \frac{1}{|\Omega|} \max_e f(e)$, i.e., the maximum flow along any edge normalized by $|\Omega|$, and the *length* $\mathcal{L}(f)$ to be the length of a longest flow-carrying path. The following theorem[5] is a special case of results in [18], which in turn generalize those in [3].

THEOREM 2.1. *For any flow $f$ on $G_\Omega$, the mixing time is bounded by $\tau_{\mathrm{mix}} \leq 4n(n+1)\mathcal{C}(f)\mathcal{L}(f)$.*

We will bound $\tau_{\mathrm{mix}}$ by constructing a flow $f$ with congestion $\mathcal{C}(f) = \mathrm{O}(n^{3/2+\epsilon})$ for any $\epsilon > 0$, and length $\mathcal{L}(f) = \mathrm{O}(n)$. By Theorem 2.1 this implies $\tau_{\mathrm{mix}} = \mathrm{O}(n^{9/2+\epsilon})$.

*Remark.* We note that our bound on the mixing time is only slightly larger than the upper bound of $\mathrm{O}(n^3)$ which one obtains by applying Theorem 2.1 to the hypercube itself (without the hyperplane constraint); see, e.g., [19]. This is in turn somewhat off from the true mixing time of $\mathrm{O}(n \log n)$. On the other hand, it is fairly easy to obtain a lower bound of $\Omega(n^2/\log n)$ for the mixing time of the truncated cube; consider, for example, an instance in which $\log n$ items have weight 1, the other $n - \log n$ items have weight $n$, and the knapsack capacity is $b = n$.

As explained in the introduction, our flow will be based on the idea of a balanced almost uniform permutation. We devote the next section to this topic and then return to the knapsack random walk in section 4.

**3. Balanced almost uniform permutations.** We begin by defining the notions of "balanced" and "almost uniform" permutations. We will write $\mathcal{S}_m$ to denote the set of all permutations of $\{1, \ldots, m\}$.

DEFINITION 3.1. *Let $\{w_i\}_{i=1}^m$ be a set of real (not necessarily positive) weights, with $M = \max_{i \leq m} |w_i|$ and $W = \sum_i w_i$, and let $\Delta \geq 1$ be a nonnegative number. A*

[4]For probability distributions $\mu, \nu$ on $\Omega$, the variation distance is defined as $\|\mu - \nu\| = \frac{1}{2}\sum_{x \in \Omega} |\mu(x) - \nu(x)| = \max_{S \subseteq \Omega} |\mu(S) - \nu(S)|$.

[5]This theorem applies to symmetric random walk on *any* connected subgraph of the hypercube $\{0,1\}^n$ in which transitions are made to each neighbor with probability $\frac{1}{2n}$.

*permutation $\pi \in \mathcal{S}_m$ is* $\Delta$-balanced *if, for all $k$ with $1 \le k \le m$,*

$$\min\{W, 0\} - \Delta M \le \sum_{i=1}^{k} w_{\pi(i)} \le \max\{W, 0\} + \Delta M. \tag{3.1}$$

Thus a balanced permutation is one whose partial sums do not fluctuate widely. In particular, if $\sum_i w_i = 0$, then condition (3.1) becomes $|\sum_{i=1}^{k} w_{\pi(i)}| \le \Delta M$.

DEFINITION 3.2. *Let $\pi$ be a random permutation in $\mathcal{S}_m$, and let $\lambda \in \mathbf{R}$. We call $\pi$ a* $\lambda$-uniform *permutation if*

$$\Pr\bigl[\pi\{1, \dots, k\} = U\bigr] \le \lambda \times \binom{m}{k}^{-1} \tag{3.2}$$

*for every $k$ with $1 \le k \le m$ and every $U \subseteq \{1, \dots, m\}$ of cardinality $k$. (Here $\pi\{1, \dots, k\}$ denotes the initial segment $\{\pi(1), \dots, \pi(k)\}$.)*

Note that, if $\pi$ were a uniform random permutation, the probability in (3.2) would be exactly $\binom{m}{k}^{-1}$ for every $U$. In a $\lambda$-uniform permutation the probabilities are permitted to vary with $U$, but only by an amount specified by the parameter $\lambda$. In our applications, $\lambda$ will be a fixed polynomial function of $m$; in this case we call $\pi$ an *almost uniform* permutation. The perhaps surprising result of this section is that, for any set of weights $\{w_i\}$, it is possible to construct an almost uniform permutation that is guaranteed to be balanced.

**3.1. The bounded ratio case.** In this section, we prove that there are balanced almost uniform permutations when the ratios of the weights are bounded by a constant. In section 3.2 we will show how to dispense with any restrictions on the weights.

LEMMA 3.3. *Let $\{w_i\}_{i=1}^{m}$ be an arbitrary set of weights with $1 \le |w_i| \le B$ for a constant $B \in [1, 2]$. Then there exists a 1-balanced permutation $\pi$ on $\{w_i\}$ which is $p(m)$-uniform, where $p(m) = Cm^{10(B-1)^2+1/2}$ for a universal constant $C$.*

*Proof.* Let $M = \max_i |w_i|$ and $W = \sum_{i=1}^{m} w_i$. Assume first that $W = 0$; we will show how to discharge this assumption later. Let $I_1 = \{i : w_i > 0\}$, $I_2 = \{i : w_i < 0\}$, $m_1 = |I_1|$, and $m_2 = |I_2|$. Define the means $\mu_1 = \frac{1}{m_1} \sum_{i \in I_1} w_i$ and $\mu_2 = -\frac{1}{m_2} \sum_{i \in I_2} w_i$. Note that $m_1\mu_1 = m_2\mu_2$ since $W = 0$.

Consider an arbitrary permutation $\nu \in \mathcal{S}_m$. This induces permutations $\nu_1, \nu_2$ on $I_1, I_2$, respectively.[6] We call $\nu_1$ $\alpha$*-good* if, for every $k_1$ with $1 \le k_1 \le m_1$,

$$\left|\sum_{i=1}^{k_1} w_{\nu_1(i)} - k_1\mu_1\right| \le \alpha(M-1)\sqrt{k_1^*}, \tag{3.3}$$

where $k_1^* = \min\{k_1, m_1 - k_1\}$, with an analogous definition for $\nu_2$. We call $\nu$ $\alpha$*-good* if both $\nu_1$ and $\nu_2$ are $\alpha$-good. Thus in a good permutation, the partial sums of both positive and negative weights are reasonably close to their expected values.

Now suppose $\nu$ is chosen u.a.r. from $\mathcal{S}_m$. A routine application of Hoeffding's bound to the partial sums (see Lemma 6.1 later) yields

$$\Pr[\nu \text{ is not } \alpha\text{-good}] \le 2m\exp(-2\alpha^2). \tag{3.4}$$

If we set $\alpha = \sqrt{\ln m}$, this probability is at most $\frac{2}{m} \le \frac{1}{2}$ for $m \ge 4$.

[6]Formally, we view $\nu_1$ as a bijection from $\{1, \dots, m_1\}$ to $I_1$, and similarly for $\nu_2$. Throughout we shall adopt this convention where appropriate, without comment.

Consider now a modified sample space in which $\nu$ is selected u.a.r. among all $\sqrt{\ln m}$-good permutations. We shall write $\Pr_{\text{unif}}$ for probabilities in the original uniform space to distinguish them from those in this modified space. By the above calculation, for any event $\mathcal{E} \subseteq \mathcal{S}_m$ we have

$$\Pr[\mathcal{E}] \le 2 \Pr_{\text{unif}}[\mathcal{E}]. \tag{3.5}$$

We are now in a position to construct our balanced almost uniform permutation. Let $\nu$ be chosen u.a.r. from all $\sqrt{\ln m}$-good permutations, and let $\nu_1, \nu_2$ be the induced permutations on $I_1, I_2$. To get a balanced permutation $\pi$, we *interleave* $\nu_1$ and $\nu_2$ as follows. We take the first element from $\nu_1$, i.e., set $\pi(1) = \nu_1(1)$. Thereafter, for each $k > 1$ in turn we set $\pi(k)$ to be the next element in $\nu_2$ if $\sum_{i=1}^{k-1} w_{\pi(i)} > 0$, and the next element in $\nu_1$ otherwise. Since $\sum_i w_i = 0$, this process is well defined and yields a permutation $\pi \in \mathcal{S}_m$. Moreover, since $|w_i| \le M$ for all $i$, it is clear that $\pi$ satisfies the balance condition (3.1) with $\Delta = 1$.

We now need to verify the uniformity condition (3.2) for $\lambda = p(m)$. Let $U \subseteq \{1, \dots, m\}$ be any subset of items with $|U| = k$ and $\Pr[\pi\{1, \dots, k\} = U] > 0$, and define

$$U_1 = U \cap I_1, \qquad U_2 = U \cap I_2, \qquad k_1 = |U_1|, \qquad k_2 = |U_2|.$$

Then we have

$$\begin{aligned} \Pr\bigl[\pi\{1, \dots, k\} = U\bigr] &\le \Pr\bigl[\nu_1\{1, \dots, k_1\} = U_1 \text{ and } \nu_2\{1, \dots, k_2\} = U_2\bigr] \\ &\le 2 \Pr_{\text{unif}}\bigl[\nu_1\{1, \dots, k_1\} = U_1 \text{ and } \nu_2\{1, \dots, k_2\} = U_2\bigr] \\ &= \frac{2}{\binom{m_1}{k_1}\binom{m_2}{k_2}}, \end{aligned} \tag{3.6}$$

where the second inequality follows from (3.5).

Now some routine calculations involving Stirling's formula (see Lemma 6.2 later) allow us to relate $\binom{m_1}{k_1}\binom{m_2}{k_2}$ to $\binom{m_1+m_2}{k_1+k_2} = \binom{m}{k}$. Specifically, (3.6) becomes

$$\Pr\bigl[\pi\{1, \dots, k\} = U\bigr] \le \frac{A m^{1/2}}{\binom{m}{k}} \exp\biggl\{ \frac{l^2 + \frac{1}{2}|l|}{\gamma(1-\gamma)} \biggl( \frac{1}{m_1} + \frac{1}{m_2} \biggr) \biggr\}, \tag{3.7}$$

where $\gamma = \frac{k}{m}$, $l = \frac{m_1 k_2 - m_2 k_1}{m}$, and $A > 0$ is a universal constant. The quantity $l$ measures the deviation of the numbers $k_1, k_2$ of positive and negative elements in $U$ from the "expected" values $\gamma m_1, \gamma m_2$, respectively. But since $\pi$ is balanced, $\nu$ is good, and the element sizes do not vary too much, $|l|$ cannot in fact be very large. To formalize this intuition, note first that

$$l = (k_2 \mu_2 - k_1 \mu_1) \frac{m_2}{\mu_1 m}, \tag{3.8}$$

since $\frac{m_2}{m_1} = \frac{\mu_1}{\mu_2}$. Now by the goodness condition (3.3) on $\nu_1, \nu_2$ we have

$$\begin{aligned} \biggl| \sum_{i=1}^{k} w_{\pi(i)} - (k_1 \mu_1 - k_2 \mu_2) \biggr| &= \biggl| \biggl( \sum_{i=1}^{k_1} w_{\nu_1(i)} + \sum_{i=1}^{k_2} w_{\nu_2(i)} \biggr) - (k_1 \mu_1 - k_2 \mu_2) \biggr| \\ &\le 2(M-1)\sqrt{k^* \ln m}, \end{aligned}$$

where $k^* = \min\{k, m-k\}$. Since $\pi$ is 1-balanced we also know that $|\sum_{i=1}^{k} w_{\pi(i)}| \le M$, and therefore

$$|k_1\mu_1 - k_2\mu_2| \le 2(M-1)\sqrt{k^* \ln m} + M.$$

Together with (3.8) and our assumption that $M \le B$, this implies the following bound on $|l|$:

$$|l| \le \Big(2(B-1)\sqrt{k^* \ln m} + B\Big)\frac{m_2}{\mu_1 m}.$$

Plugging in this value for $|l|$, we see that the exponent in (3.7) is bounded above, when $|l|$ is sufficiently large, by

$$\begin{aligned} &5(B-1)^2 k^* \ln m \frac{m_2^2}{\mu_1^2 m^2}\frac{m^2}{k(m-k)}\frac{m}{m_1 m_2} \\ &= 5(B-1)^2 \ln m \frac{k^* m}{k(m-k)}\frac{1}{\mu_1\mu_2} \\ &\le 10(B-1)^2 \ln m, \end{aligned} \tag{3.9}$$

since $k(m-k) \ge \frac{k^* m}{2}$ and $\mu_1, \mu_2 \ge 1$. Note also that if $|l|$ is bounded, then so is the exponent in (3.7), since

$$\frac{1}{\gamma(1-\gamma)}\left(\frac{1}{m_1} + \frac{1}{m_2}\right) = \frac{m^2}{k(m-k)}\frac{m}{m_1 m_2} \le \frac{m^3}{(m/2)(m/3)(m/3)} \le 18,$$

where we have used the fact that $m_1, m_2 \ge \frac{m}{3}$ which follows because $B \le 2$ and $W = 0$.

Thus (3.7) becomes

$$\Pr\big[\pi\{1,\dots,k\} = U\big] \le C\tbinom{m}{k}^{-1} m^{10(B-1)^2+1/2} \tag{3.10}$$

for a universal constant $C$, which verifies the uniformity condition (3.2) with $\lambda = p(m) = Cm^{10(B-1)^2+1/2}$.

This concludes the proof of the theorem for the case $W = \sum_i w_i = 0$. We can extend the argument to general values of $W$ using a simple trick. We will assume $W > 0$; the case $W < 0$ is entirely symmetrical. We begin by padding the sequence of weights with $d = \lceil W/M \rceil$ values $w_{m+1}, \dots, w_{m+d}$ each of which (except possibly the last) is $-M$, so that $\sum_{i=1}^{m+d} w_i = 0$. Note that $d \le m$. By the above argument for the $W = 0$ case, we can construct a 1-balanced almost uniform permutation $\pi'$ on this padded sequence (though see the remark immediately following this proof). Let $\pi$ be the *induced* permutation on the weights $\{w_i\}_{i=1}^n$. We claim that $\pi$ is also 1-balanced and almost uniform.

To see that $\pi$ is 1-balanced, note that

$$\begin{aligned} &\sum_{i=1}^{k} w_{\pi(i)} \ge \sum_{i=1}^{k'} w_{\pi'(i)} \ge -M \quad \text{and} \\ &\sum_{i=1}^{k} w_{\pi(i)} \le \sum_{i=1}^{k'} w_{\pi'(i)} + W \le M + W \end{aligned}$$

for some $k' \ge k$, using the balance property of $\pi'$.

To see that $\pi$ is almost uniform, let us call the indices $\{1,\dots,m\}$ *true* and the remainder *fake*. Let $U$ be an arbitrary subset of true indices of cardinality $k$. We need to show that

$$\Pr\bigl[\pi\{1,\dots,k\}=U\bigr]\le \tbinom{m}{k}^{-1}p(m). \tag{3.11}$$

Since $\pi$ is induced by $\pi'$, this probability is bounded above by $\sum_S \Pr[\mathcal{E}_S]$, where for $S\subseteq\{1,\dots,m+d\}$, $\mathcal{E}_S$ is the event that $\pi'\{1,\dots,|S|\}=S$ and the sum is over all $S$ of the form $U\cup U'$, where all elements of $U'$ are fake. Now by the almost uniformity of $\pi'$, this sum is at most

$$p(m+d)\sum_S \Pr_{\mathrm{unif}}[\mathcal{E}_S], \tag{3.12}$$

where $\Pr_{\mathrm{unif}}$ denotes probability under the uniform distribution on permutations in $\mathcal{S}_{m+d}$. But the sum in (3.12) is just the expectation, under the uniform distribution, of the random variable $X=\sum_S X_S$, where $X_S$ is the indicator random variable of $\mathcal{E}_S$. Thus $X$ counts the number of events $\mathcal{E}_S$ that occur. We claim that

$$\mathrm{E}(X)=\tbinom{m}{k}^{-1}\left(1+\frac{d}{m+1}\right). \tag{3.13}$$

This will complete the verification of condition (3.11), for replacing the sum in (3.12) by $\mathrm{E}(X)$ gives

$$\begin{aligned}\Pr\bigl[\pi\{1,\dots,k\}=U\bigr]&\le \tbinom{m}{k}^{-1}\left(1+\frac{d}{m+1}\right)p(m+d)\\ &\le \tbinom{m}{k}^{-1}2C(2m)^{10(B-1)^2+1/2},\end{aligned}$$

where the second inequality holds because $d\le m$. Thus, if we incorporate into $C$ an extra factor $2^{10(2-1)^2+3/2}$, we see that $\pi$ is $\lambda$-balanced for $\lambda=p(m)=Cm^{10(B-1)^2+1/2}$, as required.

To see the claim in (3.13), let $\mathcal{E}$ be the event that $\pi\{1,\dots,k\}=U$. Clearly $\Pr_{\mathrm{unif}}[\mathcal{E}]=\binom{m}{k}^{-1}$, and $X=0$ unless $\mathcal{E}$ occurs, so we have

$$\mathrm{E}(X)=\tbinom{m}{k}^{-1}\mathrm{E}(X|\mathcal{E}). \tag{3.14}$$

Conditioning now on $\mathcal{E}$, let $r$ be the position in $\pi'$ of the last element of $U$, so that $U\subseteq\pi'\{1,\dots,r\}$ and $\pi'(r)\in U$. Also, let $s$ be the position of the *next* true element; i.e., $\pi'(s)$ is true and $\pi'(t)$ is fake for $r<t<s$. (If no such element exists, let $s=m+d+1$.) Then $\mathcal{E}_S$ holds for precisely those sets $S=\pi'\{1,\dots,t\}$, where $r\le t<s$. The number of such sets is just the number of fake elements that fall between the true element at position $r$ and the next true element (at position $s$), plus one. The expectation of this quantity under the uniform distribution is plainly $1+\frac{d}{m+1}$. Plugging this into (3.14), we get the value claimed in (3.13), which concludes the proof that $\pi$ is almost uniform. ∎

*Remark.* We should point out that the padded sequence we introduced in the second part of the above proof might contain one weight whose absolute value is less than one. Thus it is not, in a strict sense, a special case of the earlier $W=0$ case, where we assumed that all the weights had absolute values in the range $[1,B]$. However, as the reader may easily verify, the analysis leading up to (3.10) still holds (with minor modifications) even when there is a single small weight.

**3.2. The general case.** In this section we extend our construction of balanced almost uniform permutations to handle arbitrary weights. The chief obstacle here is that it is no longer true (as in the bounded ratio case) that each item of positive weight can be balanced by a bounded number of items of negative weight. To overcome this difficulty, we will need to group items into "intervals" so that each interval has approximately the same (positive or negative) weight. We can then reduce to the bounded ratio case.

The following theorem is a generalization of Lemma 3.3; it says that we can construct a balanced almost uniform permutation for an arbitrary set of weights. Moreover, we can bound the uniformity parameter $\lambda$ by a polynomial whose degree is arbitrarily close to 1/2 at the cost of a modest increase in the balance parameter $\Delta$. This is almost the best that one can hope for; we encourage the reader to check by a simple counting argument that, if we have $m/2$ weights of $+1$ and $m/2$ of $-1$, then for any constants $\Delta$, $C$, and $p < 1/2$, there can be no $\Delta$-balanced $Cm^p$-uniform permutation if $m$ is sufficiently large.

In order to achieve the tightest possible bound in our random walk analysis in the next subsection, we shall actually prove a slightly stronger uniformity property which can be obtained with no additional effort. Call $\pi$ *strongly $\lambda$-uniform* if

$$\Pr\bigl[\pi\{1,\dots,k\} = U \text{ and } \pi(k+1) = l\bigr] \le \lambda \times \binom{m}{k,\ m-k-1,\ 1}^{-1} \tag{3.15}$$

for every $k$ with $1 \le k \le m$, every $U \subseteq \{1,\dots,m\}$ of cardinality $k$, and every $l \notin U$. Note that the expression on the right-hand side of (3.15) is just $\lambda$ times the probability of the given event if $\pi$ were chosen uniformly at random. Plainly (3.15) is a strengthening of (3.2) in Definition 3.2.

THEOREM 3.4. *Fix $0 < \epsilon < 2$ and let $\Delta = 1 + \sqrt{90/\epsilon}$. For any $m$ and set of weights $\{w_i\}_{i=1}^m$, there exists a $\Delta$-balanced, strongly $Cm^{1/2+\epsilon}$-uniform permutation, where $C$ is a universal constant.*

To illustrate this theorem, if we plug in the value $\epsilon = 3/2$ (say), we obtain the following corollary.

COROLLARY 3.5. *For any set of weights $\{w_i\}_{i=1}^m$, there exists a 9-balanced, $Cm^2$-uniform permutation.*

*Proof of Theorem* 3.4. First, we observe that the permutation constructed in the proof of Lemma 3.3 actually satisfies the strong uniformity property (3.15), provided that we incorporate an extra factor of 3 into the constant $C$. To see why, note that the permutation must first choose $U$ and then $l$. Suppose that the sum of the weights of the indices of $U$ is positive. Then $l$ is chosen uniformly from the remaining indices of negative weight. But these must constitute at least 1/3 of all of the remaining indices since $B \le 2$. Hence the restriction on the choice of $l$ introduces a factor of at most 3 into the uniformity parameter.

Now let $\{w_i\}$ be an arbitrary set of weights, let $M = \max_i |w_i|$, and set $\widehat{\Delta} = \frac{\Delta-1}{3}$. Let $\beta$ be a uniform random permutation in $\mathcal{S}_m$, and let $T_1$ be the smallest $t$ such that the partial sum $\sum_{i=1}^t w_{\beta(i)}$ has absolute value greater than $\widehat{\Delta}M$ (or $T_1 = m$ if no such $t$ exists). Similarly, let $T_2$ be the smallest $t > T_1$ such that $|\sum_{i=T_1+1}^t w_{\beta(i)}| > \widehat{\Delta}M$. Define $T_3, T_4, \dots$ in the same way. Then let $I_1$ be the sequence $\{\beta(i)\}_{i=1}^{T_1}$ and $I_2$ the sequence $\{\beta(T_1+i)\}_{i=1}^{T_2-T_1}$. Continue in this way, dividing $\beta$ into *intervals* $I_1, \dots, I_q$ (so that $T_q = m$).

Now let $\alpha_i$ be the aggregated weight of interval $I_i$ for $i = 1, 2, \dots, q-1$. Note that $|\alpha_i| \in [\widehat{\Delta}M, (\widehat{\Delta}+1)M]$ for all $i < q$, so the ratio of the weights of any two of these intervals is at most $(\widehat{\Delta}+1)/\widehat{\Delta} \in [1,2]$. Thus, by Lemma 3.3, there exists a 1-balanced $\lambda$-uniform permutation on $\{\alpha_i\}_{i=1}^{q-1}$ for $\lambda = Cq^{10\left((\widehat{\Delta}+1)/\widehat{\Delta}-1\right)^2+1/2} = Cq^{1/2+\epsilon}$, where $C$ is a universal constant. Moreover, as argued above, we can in fact assume that this permutation is *strongly* $\lambda$-uniform. Call this permutation $\pi_I$. We claim that the permutation

$$\pi = I_{\pi_I(1)} I_{\pi_I(2)} \cdots I_{\pi_I(q-1)} I_q$$

obtained by rearranging the first $q-1$ intervals according to $\pi_I$ is a $\Delta$-balanced strongly $Cm^{1/2+\epsilon}$-uniform permutation on the original $m$ weights.

We prove the balance property first. Let $W' = \sum_{i=1}^{q-1} \alpha_i = W - \alpha_q$. Since $\pi_I$ is 1-balanced, $\pi$ satisfies

$$\min\{0, W'\} - (\widehat{\Delta}+1)M \le \sum_{i=1}^{T_j} w_{\pi(i)} \le \max\{0, W'\} + (\widehat{\Delta}+1)M$$

for all $1 \le j \le q$. Hence we have, for all $j$,

$$\min\{0, W'\} - (2\widehat{\Delta}+1)M \le \sum_{i=1}^{j} w_{\pi(i)} \le \max\{0, W'\} + (2\widehat{\Delta}+1)M,$$

since the partial sums within any interval lie in the range $[-\widehat{\Delta}M, \widehat{\Delta}M]$. Finally, note that $|W - W'| = |\alpha_q| \le \widehat{\Delta}M$. It follows that for all $j$,

$$\min\{0, W\} - (3\widehat{\Delta}+1)M \le \sum_{i=1}^{j} w_{\pi(i)} \le \max\{0, W\} + (3\widehat{\Delta}+1)M,$$

and hence $\pi$ is $\Delta$-balanced.

To verify the strong uniformity property, consider first an alternative experiment in which the permutation $\pi_I$ is chosen u.a.r. from $\mathcal{S}_{q-1}$, without regard to the balance property. Note that, conditional on the value of $q$, the distribution of $(I_1, \dots, I_{q-1})$ is exchangeable. Thus, rearranging the intervals according to a uniform $\pi_I$ is a *measure-preserving* transformation, so $\pi$ itself has the uniform distribution. Thus we need to show that for any $U$ and any index $l \notin U$, the likelihood ratio

$$\frac{\Pr[\pi\{1,\dots,k\} = U \text{ and } \pi(k+1) = l]}{\Pr_{\text{unif}}[\pi\{1,\dots,k\} = U \text{ and } \pi(k+1) = l]} \le Cm^{1/2+\epsilon},$$

where we write $\Pr_{\text{unif}}$ for the probability when $\pi_I$ is uniform and Pr for the probability when $\pi_I$ is $Cm^{1/2+\epsilon}$-uniform. In fact, it suffices to show that the above bound on the likelihood ratio holds conditional on any $\beta$. So fix a permutation $\beta$. In order for the numerator to be nonzero, only the interval containing $l$ can contain elements from both $U$ and $U^c$ (the complement of $U$). Additionally, in the interval containing $l$, all the elements before $l$ must be in $U$ and all those after $l$ must be in $U^c$. Let $A_1$ be the collection of intervals in $\{I_i\}_{i=1}^{q-1}$ containing only elements of $U$, and let $A_2$ be the collection of intervals containing only elements of $U^c$. Then $|A_1| + |A_2|$ must have value either $q-1$ or $q-2$. Writing $\mathcal{E}_1$ for the event $\pi_I\{1, \dots, |A_1|\} = A_1$ and $\mathcal{E}_2$ for the event $\pi_I\{q-1, \dots, q-|A_2|\} = A_2$, the above likelihood ratio is

$$\frac{\Pr[\mathcal{E}_1 \text{ and } \mathcal{E}_2]}{\Pr_{\text{unif}}[\mathcal{E}_1 \text{ and } \mathcal{E}_2]} \le Cq^{1/2+\epsilon} \le Cm^{1/2+\epsilon}.$$

In the case where $|A_1| + |A_2| = q - 1$, this is just the $Cq^{1/2+\epsilon}$-uniformity property; when $|A_1| + |A_2| = q - 2$ it is the strong $Cq^{1/2+\epsilon}$-uniformity property. Thus $\pi$ is strongly $Cm^{1/2+\epsilon}$-uniform, and the proof is complete. ∎

## 4. Constructing a good flow.

We now return to the random walk for the knapsack problem and show how to construct a flow $f$ with small congestion $\mathcal{C}(f)$ and length $\mathcal{L}(f)$. Our construction will make heavy use of the balanced almost uniform permutations discussed in the previous section. In the first subsection below, we present a slightly informal high-level sketch of the argument, concentrating on showing how the balance and almost uniformity properties of the permutations are used. In section 4.2 we will give the full details of the flow.

### 4.1. High-level sketch.

Let $X, Y$ be two arbitrary vertices of $G_\Omega$, viewed as subsets of $\{1, \dots, n\}$. We need to specify how to route one unit of flow from $X$ to $Y$. The high-level idea is the following. Let $S = X \oplus Y$ (where $\oplus$ denotes symmetric difference) and $m = |S|$, and let $\{w_i\}_{i=1}^m$ be an arbitrary enumeration of the weights of the items in $S$, with the weights of items in $X, Y$ appearing with negative and positive signs, respectively. Note that each permutation of these items specifies a geodesic path from $X$ to $Y$. Thus a balanced almost uniform permutation on the $\{w_i\}$ specifies a scheme for routing unit flow between $X$ and $Y$ along geodesic paths; the amount of flow allocated to a given path is just the probability assigned to the corresponding permutation. Moreover, the weight of the $k$th point along the path is given by $a(X) + \sum_{i=1}^k w_{\pi(i)}$. We would like to use the balance property to argue that these paths remain within $\Omega$, and the almost uniformity property to argue that they are "well spread out" and thus lead to small congestion. In what follows, we assume the existence of $\Delta$-balanced, $p(m)$-uniform permutations on any set of weights $\{w_i\}_{i=1}^m$, where $\Delta$ is a fixed positive integer and $p$ is a fixed polynomial. (By Corollary 3.5 we know we can take, e.g., $\Delta = 9$ and $p(m) = \mathrm{O}(m^2)$.)

The problem with the above idea is that we need $X$ and $Y$ to be at least some small distance below the bounding hyperplane to accommodate the fluctuations that are still present within the balanced permutations. At this point, for the purposes of illustration, we make a simplifying assumption: we assume that there exists a fixed constant $h$ (independent of $n$ and the $a_i$) such that, by removing at most $h$ items from $X \oplus Y$, we arrive at vertices $X'$ and $Y'$ which satisfy $a(X'), a(Y') \le b - \Delta M$, where $M = \max_{i \in X' \oplus Y'} a_i$. Note that removing the $h = 2\Delta$ heaviest items from $X \oplus Y$ ensures this for at least one of $X', Y'$. In fact, if we increase $h$ by a constant factor, then the same holds for both $X'$ and $Y'$ for most pairs $(X, Y) \in \Omega \times \Omega$. For now, however, we simply assume this property for all pairs. Denote by $X_0, Y_0$ the sets of items removed from $X, Y$, respectively (i.e., $X = X' \cup X_0$ and $Y = Y' \cup Y_0$, with $|X_0 \cup Y_0| \le h$).

We can now describe the flow from $X$ to $Y$ in three stages:

*Stage* 1. Send the entire unit flow along a single path from $X$ to $X'$ by removing the items in $X_0$ in index order.

*Stage* 2. Distribute the unit flow along geodesic paths from $X'$ to $Y'$ according to a balanced almost uniform permutation $\pi$ on the (signed) weights $\{w_i\}_{i=1}^m$ of the items in $S = X' \oplus Y'$, where $m = |S|$.

*Stage* 3. Send the entire unit flow along a single path from $Y'$ to $Y$ by adding the items in $Y_0$ in index order.

Figure 4.1 gives a schematic illustration of this flow. Notice how the upper bound in the balance property keeps the path below the hyperplane, while the lower bound

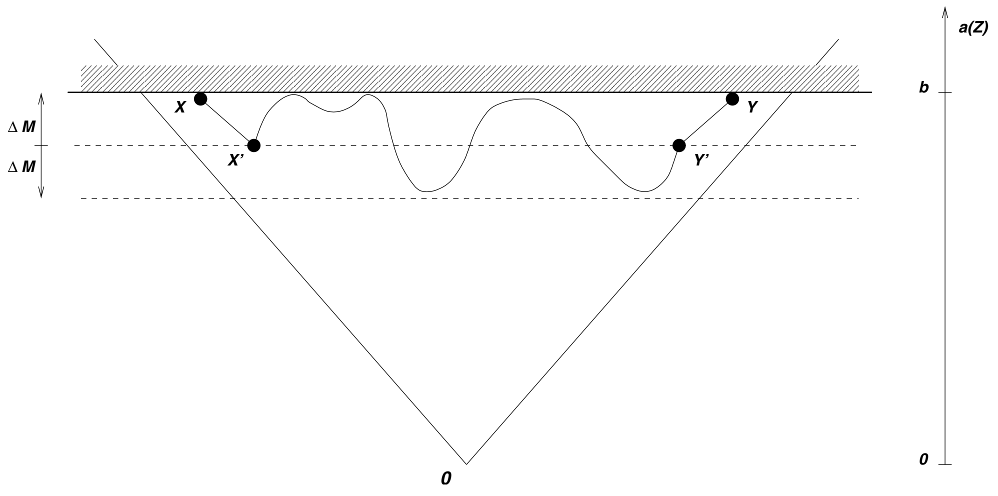


FIG. 4.1. *Schematic illustration of flow from $X$ to $Y$. The function $a(\cdot)$ increases in the vertical direction; the bounding hyperplane is $a(Z) = b$. In the case shown, $a(X') = a(Y')$; the balance property of the permutation of $X' \oplus Y'$ ensures that the path remains within $\pm\Delta M$ of this level.*

keeps the path from passing too close to the origin (thus avoiding a potential bottleneck).

Let us first observe that the above flow is valid. For this, we just need to check that all the flow-carrying paths remain within the set $\Omega$. This is obvious for stages 1 and 3. For stage 2 it follows from the balance property of $\pi$; for if $Z$ is the $k$th point along a flow-carrying path from $X'$ to $Y'$, then

$$\begin{aligned} a(Z) &= a(X') + \sum_{i=1}^{k} w_{\pi(i)} \\ &\le a(X') + \max\{a(Y') - a(X'), 0\} + \Delta M \\ &= \max\{a(Y'), a(X')\} + \Delta M \\ &\le b, \end{aligned} \tag{4.1}$$

where in the last line we have used the fact that $a(X'), a(Y') \le b - \Delta M$. Hence $Z \in \Omega$.

Next we must bound the quantities $\mathcal{C}(f)$ and $\mathcal{L}(f)$ for this flow $f$, as defined in section 2. $\mathcal{L}(f)$, the length of a longest flow-carrying path, is plainly at most $m \le n$. To estimate the congestion $\mathcal{C}(f)$, we must bound the flow along any edge of $G_\Omega$. For convenience in this sketch, we will in fact bound the flow $f(Z)$ through any *vertex* $Z$; clearly this is also an upper bound on the flow along any edge.

So let $Z$ be an arbitrary vertex of $G_\Omega$. Define $\mathcal{P}(Z)$ to be the set of pairs $(X, Y)$ such that some $X \to Y$ flow passes through $Z$. Note that $\mathcal{P}(Z) = \bigcup_{i=1}^{3} \mathcal{P}_i(Z)$, where $\mathcal{P}_i(Z)$ are the pairs whose paths pass through $Z$ in stage $i$. We shall bound the contribution to $f(Z)$ from each $\mathcal{P}_i(Z)$ separately. For $i = 1, 3$ this is simple: since stage-1 paths have length at most $h$, for any given $Y \in G_\Omega$ the number of vertices $X$ such that $(X, Y) \in \mathcal{P}_1(Z)$ is (crudely) at most $(h+1)n^h$, so the contribution to $f(Z)$ from such paths is no more than $(h+1)n^h|\Omega|$. The same bound holds symmetrically for $\mathcal{P}_3(Z)$. The main portion of the paths, $\mathcal{P}_2(Z)$, presents more of a challenge.

We shall actually work with $\mathcal{P}_2'(Z)$, the set of pairs $(X',Y')$ such that $Z$ lies on the stage-2 path with endpoints $X',Y'$. By the observation in the previous paragraph, the flow contribution from $\mathcal{P}_2(Z)$ will be (again crudely) at most $(h+1)^2n^{2h}$ times that from $\mathcal{P}_2'(Z)$. Recall that we are really interested in the ratio $\frac{f(Z)}{|\Omega|}$, rather than in $f(Z)$ itself. Accordingly, following earlier analyses of this general type (see, e.g., [10, 11]), we measure the set $\mathcal{P}_2'(Z)$ by associating with each of its elements $(X',Y')$ an "encoding" $\widetilde{Z}$, which belongs to $\Omega$. This is defined by

$$\widetilde{Z} = X' \oplus Y' \oplus Z.$$

To see that $\widetilde{Z} \in \Omega$, we need to check that $a(\widetilde{Z}) \le b$. But this follows because

$$\begin{aligned} a(\widetilde{Z}) &= a(X') + a(Y') - a(Z) \\ &\le a(X') + a(Y') - t(\min\{a(X'), a(Y')\} - \Delta M) \\ &= \max\{a(X'), a(Y')\} + \Delta M \\ &\le b, \end{aligned}$$

where in the second line we have used the balance property of $\pi$ as in (4.1) to bound $a(Z)$, this time from below.

How many pairs $(X',Y')$ could be mapped to a given $\widetilde{Z}$? First note that $\widetilde{Z}$ uniquely determines both $S = X' \oplus Y'$ and $I = X' \cap Y'$ via the relations

$$S = \widetilde{Z} \oplus Z, \qquad I = \widetilde{Z} \cap Z.$$

Thus, in particular, such pairs share the same symmetric difference, $S$, of cardinality $m$, say. To determine $X'$ and $Y'$ uniquely, it suffices to specify the subset $U \subseteq S$ of elements that have already been processed (i.e., added or deleted) by the stage-2 path by the time it reaches $Z$. For then we know, from the fact that all stage-2 paths are geodesic, that $X'$ agrees with $Z$ on $S-U$ and with $\widetilde{Z}$ on $U$, and vice versa for $Y'$. More formally,

$$X' = Z \oplus U, \qquad Y' = \widetilde{Z} \oplus U.$$

The upshot of the foregoing discussion is that we can define a mapping from $\mathcal{P}_2'(Z)$ to pairs of the form $(\widetilde{Z},U)$, where $\widetilde{Z} \in \Omega$ and $U$ is a subset of $Z \oplus \widetilde{Z}$. Moreover, and crucially, this mapping is *injective*. It therefore effectively enumerates the set $\mathcal{P}_2'(Z)$.

Finally, we need to take account of the actual quantity of flow traveling along the paths. Consider a pair $(X',Y') \in \mathcal{P}_2'(Z)$, corresponding to the pair $(\widetilde{Z},U)$. Recall that the flow distribution between $X'$ and $Y'$ is determined by a balanced almost uniform permutation $\pi$ of the weights in $S = X' \oplus Y'$. The proportion of this flow that passes through $Z$ is precisely

$$\Pr\bigl[\pi\{1,\dots,|U|\} = U\bigr] \le \tbinom{m}{|U|}^{-1} p(m),$$

by the almost uniform property of $\pi$.

Putting all this together, we can bound the total contribution to $f(Z)$ from $\mathcal{P}_2'(Z)$ as follows:

$$\begin{aligned} \sum_{\widetilde{Z}\in\Omega} \sum_{U\subseteq Z\oplus\widetilde{Z}} \Pr\bigl[\pi\{1,\dots,|U|\} = U\bigr] &\le \sum_{\widetilde{Z}\in\Omega} \sum_{k} \sum_{U\subseteq Z\oplus\widetilde{Z},\,|U|=k} \tbinom{m}{k}^{-1} p(m) \\ &\le p(n) \sum_{\widetilde{Z}\in\Omega} \sum_{k} \tbinom{m}{k}\tbinom{m}{k}^{-1} \\ &\le np(n)|\Omega|, \end{aligned}$$

where in the summations $m = |Z \oplus \widetilde{Z}|$. The total contribution from all stage-2 paths is thus at most $(h+1)^2 n^{2h+1} p(n)|\Omega|$.

Combining this with our earlier bounds for stages 1 and 3, we obtain that $f(Z) \leq (2(h+1)n^h + (h+1)^2 n^{2h+1} p(n))|\Omega|$, and hence $\mathcal{C}(f) \leq p'(n)$ (for a different polynomial $p'$). Since both $\mathcal{L}(f)$ and $\mathcal{C}(f)$ are bounded polynomially in $n$, we obtain immediately from Theorem 2.1 that the mixing time, $\tau_{\text{mix}}$, is polynomial in $n$ as required.

In the above sketch, we assumed that we could remove a fixed number of items from $X$ and $Y$ and thereby ensure that they are well separated from the bounding hyperplane. The main extra work in the full description of the flow in the next subsection is concerned with discharging this assumption. In addition, we will be more precise in our accounting so as to achieve the tightest possible bound on $\mathcal{C}(f)$.

**4.2. Complete description.** In this subsection, we present a complete construction, for an arbitrary instance of the 0-1 knapsack problem, of a flow $f$ with congestion $\mathcal{C}(f) = \mathrm{O}(n^{3/2+\epsilon})$ for any $\epsilon > 0$ and length $\mathcal{L}(f) = \mathrm{O}(n)$. By Theorem 2.1, this will imply that the mixing time of the random walk on $G_\Omega$ is $\mathrm{O}(n^{9/2+\epsilon})$. From now on we assume that $\epsilon > 0$ is arbitrary but fixed.

Let $X, Y$ be arbitrary vertices of $G_\Omega$. Recall the scheme for constructing a flow from $X$ to $Y$ in the sketch of section 4.1; we essentially followed a balanced almost uniform permutation of $X \oplus Y$, except that we removed a constant number of items from consideration (processing them at the beginning and end of the path) to ensure that both path endpoints were far enough from the hyperplane so that the path remained within $G_\Omega$. In reality this property is not guaranteed by the removal of a fixed number of items because of the possibly large variations in weights between $X$ and $Y$. (For example, if all the items in $X$ are much larger than those in $Y$, then removing any fixed number of items from $X$ and $Y$ could still leave $Y$ too close to the hyperplane relative to the size of the largest remaining item in $X \oplus Y$.) However, it turns out that after removing a fixed number of heavy items from $X \oplus Y$, we can perform a "preprocessing" operation by randomly switching items between $X$ and $Y$ to roughly balance their weights. Moreover, we will need to add and delete the removed items repeatedly along the path to maintain fine balance. The resulting flow-carrying paths will not in general be geodesics, as before, though they will have length only $\mathrm{O}(n)$.

In preparation for describing the flow, we first describe the preprocessing operation. We assume that $a(X) + a(Y) \leq 2b - 6\Delta M$, where $M = \max_{i \in X \oplus Y} a_i$ and $\Delta = \Delta(\epsilon)$ is the constant appearing in Theorem 3.4. (We will reduce to this case by first deleting a fixed number of items from $X \oplus Y$.) Call the pair $(X, Y)$ *full* if either $a(X) > b - \Delta M$ or $a(Y) > b - \Delta M$. Our goal is to shift items randomly between $X$ and $Y$ and thereby reach a pair $(X', Y')$ that is not full.

Consider the following random walk on $\{(X', Y') : X' \cup Y' = X \cup Y,\ X' \cap Y' = X \cap Y,\ a(X'), a(Y') \leq b\}$. If the current state is $(X', Y')$, choose an index $i \in X' \oplus Y'$ u.a.r. With probability $\frac{1}{2}$, do nothing; else move $i$ from $X'$ to $Y'$ or $Y'$ to $X'$ if possible. We call this the *preprocessing random walk* (PRW). We claim in the following lemma that, if we run the PRW for a number of steps chosen randomly between 1 and $\mathrm{O}(n)$, we will with reasonable probability arrive at a pair $(X', Y')$ that is not full. The proof uses a martingale argument and is deferred to section 6.

LEMMA 4.1. *Let $(X, Y)$ be a full pair of vertices in $G_\Omega$ with $a(X) + a(Y) \leq 2b - 6\Delta M$, where $M = \max_{i \in X \oplus Y} a_i$. Pick $T$ u.a.r. from $\{1, 2, \ldots, C_1 m\}$, where $m = |X \oplus Y|$ and $C_1$ is a suitable constant (which depends only on $\Delta$), and let*

*$(X', Y')$ be the result of running the PRW for $T$ steps starting from $(X, Y)$. Then* $\Pr[(X', Y')$ *is not full*$] \geq 1/C_2$ *for a positive constant $C_2$ (which again depends only on $\Delta$).*

We are now ready to construct and analyze the flow.

LEMMA 4.2. *For arbitrary weights and any $\epsilon > 0$, it is possible to construct a multicommodity flow $f$ in $G_\Omega$ with $\mathcal{C}(f) = \mathrm{O}(n^{3/2+\epsilon})$ and $\mathcal{L}(f) = \mathrm{O}(n)$.*

*Proof.* Let $X, Y$ be arbitrary vertices of $G_\Omega$. Viewing $X$ and $Y$ as subsets of $\{1, \ldots, n\}$, let $H$ be the $h = \lceil 6\Delta \rceil$ elements of $X \oplus Y$ having largest weight (or let $H = X \oplus Y$ if $|X \oplus Y| \leq h$), with ties broken according to index order. Define $X' = X - H$, $Y' = Y - H$, and $S = X' \oplus Y'$. Let $m = |S|$ and $M = \max_{i \in S} a_i$.

We will say that a set of indices $Z$ is *good* if $Z - H \in \Omega$ and $(Z \oplus X \oplus Y) - H \in \Omega$. For a set of indices $Z$ and an index $i$, define

$$Zi = \begin{cases} Z \oplus \{i\} & \text{if } Z \oplus \{i\} \text{ is good,} \\ Z & \text{otherwise.} \end{cases}$$

Define $Zi_1i_2 = ((Zi_1)i_2)$ and so on. Note that if $Y = Xi_1 \cdots i_l$, then the sequence $i_1, \ldots, i_l$ defines a path from $X$ to $Y$ in the unit hypercube of length at most $l$. This path need not in general lie within $G_\Omega$; however, it is "close to" $G_\Omega$ in the sense that for every point $Z$ of the path, $Z - H \in \Omega$.

If $(X', Y')$ is not full, set $T = 0$; otherwise, choose $T$ u.a.r. from $\{1, \ldots, C_1 m\}$, where $C_1$ is the constant in Lemma 4.1. Next, let $i_1, \ldots, i_T$ be chosen independently from the uniform distribution over $S$. Define $X'' = X'i_1 \cdots i_T$ and $Y'' = X'' \oplus X' \oplus Y' = Y'i_1 \cdots i_T$. Thus $(X'', Y'')$ is the result of running the PRW for $T$ steps starting from $(X', Y')$. Note that $a(X') + a(Y') \leq 2b - a(H) \leq 2b - 6\Delta M$. So, by Lemma 4.1, we can condition on the event that the pair $(X'', Y'')$ is not full and thus increase the probability of any path by a factor of at most $C_2$.

Now let $\{w_i\}_{i=1}^m$ be an arbitrary enumeration of the weights of the items in $S$, with the weights of items in $X'', Y''$ appearing with negative and positive signs, respectively, and let $\pi$ be a $\Delta$-balanced, strongly $Cm^{1/2+\epsilon}$-uniform permutation on the $\{w_i\}$ whose existence is guaranteed by Theorem 3.4. We claim that the sequence

$$i_1, \ldots, i_T, \pi(1), \ldots, \pi(m), i_T \ldots, i_1 \tag{4.2}$$

defines a path from $X$ to $Y$ in the hypercube. This is true because the condition that $(X'', Y'')$ be not full, together with the fact that $\pi$ is balanced, guarantees that all of the transitions indicated by $\pi$ will actually take place.

Set

$$j_k = \begin{cases} i_k & \text{if } 1 \leq k \leq T, \\ \pi(k - T) & \text{if } T < k \leq T + m, \\ i_{2T+m-k-1} & \text{if } T + m < k \leq 2T + m, \end{cases}$$

and let $l = 2T + m$. Then $j_1, \ldots, j_l$ is the sequence in (4.2). Our flow from $X$ to $Y$ will essentially follow the sequence $j_k$, except that along the way elements of $H$ will be used to keep the knapsack as full as possible but will be removed as necessary to make room for new items $j_k$ to be added. Thus each intermediate state $Z$ will be of the form $\overline{H} \oplus Xj_1 \cdots j_k$, for some $\overline{H} \subseteq H$ and $k \leq l$.

Suppose that, after processing the first $k \leq l$ elements of the sequence in (4.2), we have $Z = \overline{H} \oplus Xj_1 \cdots j_k$ for some $\overline{H} \subseteq H$. The transition rule will be as follows.

1. If $k < l$ and $j_{k+1} \notin Z$, then move to $Zj_{k+1}$ if possible (i.e., if the result is an element of $\Omega$); otherwise, delete an element from $H$.

2. If $k < l$ and $j_{k+1} \in Z$, then add an element from $H$ if possible; otherwise, move to $Zj_{k+1}$.
3. If $k = l$, then add an element from $H \cap Y$ if possible; otherwise, delete an element from $H \cap X$.

The fact that all of the sets $Xj_1 \cdots j_k$ are good ensures that sufficient elements of $H$ can always be removed so as to make room to add the next element $j_{k+1}$ when necessary; hence the above rule defines a feasible random path from $X$ to $Y$. Similarly, goodness also implies that $a(Z \oplus X \oplus Y - H) \le b$ for every intermediate state $Z$; since our rule keeps the weight as large as possible, this implies that, at any intermediate edge $(Z, W)$ along the path, there exists (at most) one element $u \in H$ such that

$$a(Z \oplus X \oplus Y - \{u, z\}) \le b, \tag{4.3}$$

where $z$ is the index such that $\{z\} = Z \oplus W$. Then $(\widetilde{Z} - \{u, z\}) \in \Omega$, where, exactly as in the analysis in section 4.1, we define the "encoding" $\widetilde{Z}$ by

$$\widetilde{Z} = X \oplus Y \oplus Z.$$

Thus, for any given edge $(Z, W)$, the number of encodings $\widetilde{Z}$ is at most $n|\Omega|$.

Note that the path from $X$ to $Y$ can be naturally divided into three *stages*, corresponding to the three parts of the sequence $j_k$. We will write the flow through any given edge $(Z, W) \in G_\Omega$ as $f(Z, W) = f_1(Z, W) + f_2(Z, W) + f_3(Z, W)$, where $f_i(Z, W)$ is the contribution of stage-$i$ paths. We will bound $f$ by bounding each of the three contributions $f_i$ separately.

Consider stage 1 first, and focus on a particular edge $(Z, W)$. For any pair $(X, Y)$ that sends flow through $(Z, W)$ in stage 1, we can write $Z = \overline{H} \oplus Xj_1 \cdots j_k$, where $j_1, \dots, j_k$ are the first $k$ elements processed along the path. Thus the pair $(X, Y)$ is completely specified by $k$, $j_1, \dots, j_k$, $\widetilde{Z}$, and $\overline{H}$, via the easily verified relations

$$X = \overline{H} \oplus Zj_k \cdots j_1, \qquad Y = \overline{H} \oplus \widetilde{Z}j_k \cdots j_1.$$

The amount of flow corresponding to a given sequence $j_1, \dots, j_k$ is bounded above by the probability that $j_1, \dots, j_k, z$ were the first $k+1$ indices chosen in the PRW, which is at most $C_2 m^{-(k+1)}$. (The factor $C_2$ here arises from our earlier conditioning on the event that $(X'', Y'')$ is not full.) Thus we can bound the stage-1 flow $f_1(Z, W)$ as in section 4.1. We have

$$\begin{aligned} f_1(Z, W) &\le \sum_{\widetilde{Z}} \sum_{k} \sum_{j_1, \dots, j_k} \sum_{\overline{H}} C_2 m^{-(k+1)} \\ &\le \sum_{\widetilde{Z}} \sum_{k} 2^h C_2 m^{-1} \\ &\le \sum_{\widetilde{Z}} C_1 m 2^h C_2 m^{-1} \\ &\le C_1 C_2 2^h n|\Omega|, \end{aligned}$$

where the factors $C_1 m$ and $2^h$ arise from summing over $k$ and $\overline{H}$, respectively.

The flow $f_3(Z, W)$ from stage-3 paths can be handled symmetrically, so consider now the stage-2 paths. For a given edge $(Z, W)$, every pair $(X, Y)$ that sends flow through $(Z, W)$ in stage 2 can be completely specified by $\widetilde{Z}$, $T$, $k$, $j_1, \dots, j_T$, $U$, and $\overline{H}$, where $k$ is the number of elements of the sequence in (4.2) processed along the path from $X$ to $Z$ and $U = \{\pi(1), \ldots, \pi(k-T)\}$, via

$$X = \overline{H} \oplus (Z \oplus U)j_T \cdots j_1, \qquad Y = \overline{H} \oplus (\widetilde{Z} \oplus U)j_T \cdots j_1.$$

Let $k' = k - T$. The amount of flow corresponding to a given $j_1, \ldots, j_T$ and $U$ is bounded above by

$$(C_2 m^{-T})(C_1 m)^{-1}\left[Cm^{1/2+\epsilon}\tbinom{m}{k', m-k'-1, 1}^{-1}\right],$$

where the first factor comes from the PRW, the second factor is the probability of choosing a particular $T$, and the third factor is an upper bound on the probability $\Pr[\pi\{1, \ldots, k'\} = U \text{ and } \pi(k'+1) = z]$, which comes from the strong almost uniformity of $\pi$. Thus we can again bound the flow $f_2(Z, W)$ as in section 4.1. We have

$$\begin{aligned} f_2(Z,W) &\le \sum_{\widetilde{Z}} \sum_{T} \sum_{k} \sum_{j_1,\ldots,j_T} \sum_{U} \sum_{\overline{H}} (C_2 m^{-T})(C_1 m)^{-1}\left[Cm^{1/2+\epsilon}\tbinom{m}{k', m-k'-1, 1}^{-1}\right] \\ &\le \sum_{\widetilde{Z}} (C_1 m) m m^T \tbinom{m-1}{k'} 2^h (C_2 m^{-T})(C_1 m)^{-1}\left[Cm^{1/2+\epsilon}\tbinom{m}{k', m-k'-1, 1}^{-1}\right] \\ &= \sum_{\widetilde{Z}} 2^h C_2 \left[m\tbinom{m-1}{k'}\tbinom{m}{k', m-k'-1, 1}^{-1}\right] Cm^{1/2+\epsilon} \\ &= \sum_{\widetilde{Z}} 2^h C_2 C m^{1/2+\epsilon} \\ &\le 2^h C C_2 n^{3/2+\epsilon} |\Omega|, \end{aligned}$$

where the factors in the second line are written in the same order as the sums they arise from.

Adding the contributions $f_1$, $f_2$, and $f_3$, we see that the above flow satisfies $\mathcal{C}(f) = \mathrm{O}(n^{3/2+\epsilon})$, while plainly $\mathcal{L}(f) = \mathrm{O}(n)$. Since $\epsilon > 0$ was arbitrary, this completes the proof. $\square$

Given such a flow, we need only invoke Theorem 2.1 to derive our main result.

THEOREM 4.3. *Let $\Omega$ be the set of solutions to an arbitrary instance of the* 0-1 *knapsack problem. The mixing time of the random walk on $G_\Omega$ is $\tau_{\text{mix}} = \mathrm{O}(n^{9/2+\epsilon})$ for any $\epsilon > 0$.*

As mentioned in the introduction, this immediately yields an fpras for computing $|\Omega|$, via a standard reduction to random sampling (whose details are spelled out in [12]).

*Remark.* The mixing time bound of $\mathrm{O}(n^{9/2+\epsilon})$ in Theorem 4.3 is reasonably tight for this type of analysis. If we apply Theorem 2.1 to analyze random walk on the entire cube $\{0,1\}^n$, we get a bound of $\mathrm{O}(n^3)$ even with an optimal flow. Thus the truncation introduces an extra factor of only $\mathrm{O}(n^{3/2+\epsilon})$ into the bound. It is instructive to see where this extra factor comes from; $\mathrm{O}(n^{1/2+\epsilon})$ is due to the balanced almost uniform permutation construction (Theorem 3.4, which is tight), while $\mathrm{O}(n)$ comes from the fact that the "encoding" $\widetilde{Z}$ may lie just outside $\Omega$.

## 5. Multiple hyperplanes.

### 5.1. Introduction.

In this section, we will extend our earlier results to handle multiple hyperplanes. For a nonnegative real $d \times n$ matrix $A = (a_{ij})$ and a positive real vector $\mathbf{b} = (b^1, \ldots, b^d)$, let $\Omega$ denote the set of 0-1 vectors $\mathbf{x} = (x_i)_{i=1}^n$ for which $A\mathbf{x} \leq \mathbf{b}$. The vertices in $\Omega$ constitute the set of feasible solutions to the multidimensional knapsack problem with the $d$ simultaneous constraints

$$\mathbf{a}^j \cdot \mathbf{x} \equiv \sum_{i=1}^n a_i^j x_i \leq b^j \quad \text{for } 1 \leq j \leq d, \tag{5.1}$$

where $a_i^j \equiv a_{ji}$. (In (5.1) the superscript $j$ indexes the $j$th linear constraint; we will follow this convention throughout.)

Geometrically, $\Omega$ is obtained by truncating the unit cube by $d$ hyperplanes, each of which corresponds to a knapsack constraint. The essential geometric property of these "knapsack" hyperplanes is that their normal vectors all lie in the same quadrant. The results of this section will easily extend to any collection of hyperplanes with this property.[7]

Following our earlier notation, we identify a 0-1 vector $\mathbf{x} = (x_i)_{i=1}^n$ with the set of indices $X = \{i : x_i = 1\}$ and write $a(X) = (a^1(X), \ldots, a^d(X))$ for the (now $d$-dimensional) weight of $X$. As before we denote by $G_\Omega$ the subgraph of the hypercube $\{0,1\}^n$ induced by the vertices in $\Omega$, and we again study symmetric random walk on $G_\Omega$; i.e., transitions from a given state $X \subseteq \{1, \ldots, n\}$ are made as follows:

1. pick an item $i \in \{1, \ldots, n\}$ u.a.r.;
2. if $i \in X$, move to $X - \{i\}$; if $i \notin X$ and $a^j(X \cup \{i\}) \leq b^j$ for all $j$, move to $X \cup \{i\}$; otherwise, do nothing.

Again, to avoid issues involving periodicity, we add to every state a holding probability of $\frac{1}{2}$.

In this section we will prove that this random walk on $G_\Omega$ has mixing time that is polynomially bounded in $n$ for any fixed dimension $d$. Just as in the one-dimensional case, this immediately gives a polynomial time algorithm for sampling (almost) uniformly at random from $\Omega$ and an fpras for computing $|\Omega|$.

We note that the degree of our polynomial upper bound for the mixing time will depend on the dimension $d$, but this is unavoidable as the following simple example shows. Consider a $d$-dimensional knapsack problem in which there are $\frac{n}{2d}$ items having each of the $d$ weight vectors $(n, 0, \ldots, 0), (0, n, \ldots, 0), \ldots, (0, \ldots, 0, n)$, and the remaining $\frac{n}{2}$ items have weight vector $(1, 1, \ldots, 1)$; the knapsack capacity is $\mathbf{b} = (n, \ldots, n)$. Let $S$ be the set of feasible solutions in $\Omega$ which do not contain any of the $(1, \ldots, 1)$ items. Then $|S| = (\frac{n}{2d} + 1)^d$, but $S$ is connected to $\Omega - S$ only through the origin. It follows easily that the mixing time is $n^{\Omega(d)}$.

In fact, for arbitrary $d$ there can be no uniform polynomial upper bound for the running time of *any* sampling algorithm unless RP = NP. This follows immediately by reduction from the problem of sampling independent sets in a graph. By Theorem 1.17 of [19], there is no algorithm for (almost) uniformly sampling independent sets in a graph unless RP = NP. Now if $G = (V, E)$ is an arbitrary (undirected) graph, there is a 1-1 correspondence between the independent sets in $G$ and the feasible solutions to the knapsack problem with $|V|$ variables and the $|E|$ constraints $x_u + x_v \leq 1$ for all $\{u, v\} \in E$.

To prove rapid mixing of the random walk on $G_\Omega$ for any fixed $d$, we use the multicommodity flow technique as before. Recall that Theorem 2.1, which bounds

[7]However, we cannot allow the hyperplanes to be arbitrary. If arbitrary truncations were allowed, then it would be possible to use just two hyperplanes to cause exponential bottlenecks or even disconnect the graph $G_\Omega$.

the mixing time in terms of the cost of a flow $f$, holds for symmetric random walk on any connected subset of the hypercube, so it again suffices to come up with a flow of small cost. As before, the idea is to spread each $X \to Y$ flow evenly using a balanced almost uniform permutation. However, since the weight function $a(\,\cdot\,)$ is now vector-valued, we first need to extend the definition of balance to higher dimensions.

DEFINITION 5.1. *Fix an integer $d > 0$, and let $\{w_i\}_{i=1}^m$ be a set of weights in $\mathbf{R}^d$ satisfying $\sum_{i=1}^m w_i = 0$. For a positive real number $\Delta$, a permutation $\pi \in \mathcal{S}_m$ is* $\Delta$-balanced *if*

$$\max_k \left| \sum_{i=1}^k w^j_{\pi(i)} \right| \le \Delta M^j \quad \textit{for } 1 \le j \le d, \tag{5.2}$$

*where $w_i = (w_i^1, \dots, w_i^d)$ and $M^j = \max_{1\le i\le m} |w_i^j|$.*

Thus $\pi$ is balanced with respect to vector weights $\{w_i\}$ if and only if it satisfies the $d$ one-dimensional balance conditions given by (5.2). Note that this generalizes Definition 3.1 for the one-dimensional case (except that, for simplicity, we have assumed that $\sum_i w_i = 0$).

Constructing balanced almost uniform permutations is significantly more difficult in higher dimensions since one has to control fluctuations in all dimensions simultaneously. In fact, for $d \ge 2$, it is nontrivial to prove for an arbitrary set of vector weights that even a *single* balanced permutation exists. (For $d = 1$, of course, this is trivial.) The existence of such a permutation follows at once from a lemma due to Grinberg and Sevast'yanov [7], which was proved in an entirely different context.

LEMMA 5.2 (see [7]). *Let $x_1, \dots, x_n$ be vectors in $\mathbf{R}^d$ such that $\sum_i x_i = 0$. Then there exists a permutation $\nu \in \mathcal{S}_n$ such that*

$$\sum_{i=1}^k x_{\nu(i)} \in d \times \operatorname{conv}\{x_1, \dots, x_n\} \quad \textit{for } 1 \le k \le n,$$

*where $d \times \operatorname{conv}$ denotes the dilation of the convex hull by $d$.*

Of course, we need something much stronger than this, namely, *almost uniform* permutations with a similar balance property. Perhaps surprisingly, we will show that balanced almost uniform permutations exist in arbitrary dimension $d$. To illustrate the main ideas involved in extending from one to higher dimensions, we now give a sketch of the proof in the special case where $d = 2$ and the weights satisfy $1 \le |w_i^j| \le 2$ for all $i$ and $j$.

In this setting, let $I_1 = \{i : w_i^2 \ge 0\}$, $I_2 = \{i : w_i^2 < 0\}$, and define $v = \sum_{i \in I_1} w_i$. For every $i \le m$, let $y_i$ be the projection of $w_i$ onto $v^\perp$. Let $\pi_1$ be an almost uniform permutation on $I_1$ which is balanced (in the one-dimensional sense) with respect to $\{y_i\}_{i\in I_1}$, with a similar definition for $\pi_2$. Finally, interleave $\pi_1$ and $\pi_2$ to give a permutation on $\{1, \dots, m\}$ which is balanced with respect to $\{w_i^2\}_{i=1}^m$ (the projections of the $w_i$ onto the second coordinate axis). Since $\pi_1$ and $\pi_2$ are both almost uniform, so is $\pi$, by an argument similar to that in the proof of Lemma 3.3.

Furthermore, since $\pi_1$ and $\pi_2$ are each balanced with respect to projections onto $v^\perp$, so is $\pi$. (Note that the projections $y_i$ satisfy $\sum_{i\in I_1} y_i = \sum_{i \in I_2} y_i = 0$.) Thus, for every $k$, the projections of $\sum_{i=1}^k w_{\pi(i)}$ onto the second coordinate axis and onto $v^\perp$ are both bounded, and since the $w_i^j$ are all in $[1, 2]$, the angle between the coordinate axis and $v^\perp$ is bounded away from zero. Thus, the partial sums $\sum_{i=1}^k w_{\pi(i)}$

stay inside a parallelogram of bounded diameter. Hence $\pi$ is balanced with respect to the weights $\{w_i\}_{i=1}^m$.

This concludes the sketch proof for the above special case with $d = 2$. Note that it is a straightforward reduction to the one-dimensional result. Unfortunately, in general the reduction from $d$ to $d-1$ dimensions is not quite so straightforward; we deal with the extra technical difficulties in the next subsection.

**5.2. Balanced almost uniform permutations in arbitrary dimensions.** The following theorem says that one can always construct balanced almost uniform permutations when the dimension $d$ is fixed.

THEOREM 5.3. *Let $d$ be any positive integer. There is a constant $c_d$ and a polynomial function $p_d$ such that, for any set of weights $\{w_i\}_{i=1}^m$ in $\mathbf{R}^d$ with $\sum_i w_i = 0$, there exists a $c_d$-balanced, $p_d(m)$-uniform permutation.*

*Proof.* The proof will be by induction on $d$. The base case $d = 1$ follows from Corollary 3.5, with $c_1 = 9$ and $p_1(m) = Cm^2$. Now let $d > 1$ be arbitrary, and suppose that the result holds for dimensions up to $d-1$. Let $\{w_i\}_{i=1}^m$ be a set of weights in $\mathbf{R}^d$. Suppose first that the weights satisfy

$$M^j = 2 \qquad \text{for all } j, \tag{5.3}$$

$$1 \le \max_{1\le j\le d} |w_i^j| \le 2 \qquad \text{for all } i. \tag{5.4}$$

Thus each weight is at least half as large as the maximum (positive or negative) weight in some coordinate. Then

$$\max_{1\le j\le d} \sum_{i=1}^m |w_i^j| \ge \frac{m}{d}.$$

Without loss of generality, suppose that the sum in the left-hand side is maximized by $j = d$. Then we have

$$\sum_{i=1}^m (w_i^d)^+ = \sum_{i=1}^m (w_i^d)^- \ge \frac{m}{2d},$$

where we are using the notation $z^+ = \max\{z, 0\}$ and $z^- = \max\{-z, 0\}$.

Let $I_1 = \{i : w_i^d \ge 0\}$, $I_2 = \{i : w_i^d < 0\}$, $m_1 = |I_1|$, and $m_2 = |I_2|$. Define the means $\mu_1 = \frac{1}{m_1}\sum_{i\in I_1} w_i^d$ and $\mu_2 = -\frac{1}{m_2}\sum_{i\in I_2} w_i^d$. Note that $\mu_1, \mu_2 \ge \frac{1}{2d}$. For $1 \le j < d$, let

$$\gamma^j = \frac{\sum_{i\in I_1} w_i^j}{\sum_{i\in I_1} w_i^d} = \frac{\sum_{i\in I_2} w_i^j}{\sum_{i\in I_2} w_i^d}.$$

For all $i \le m$ and $j < d$, let $y_i^j = w_i^j - \gamma^j w_i^d$, and let $y_i = (y_i^1, \dots, y_i^{d-1})$. Note that $|\gamma^j| \le 1$, $|y_i^j| \le 4$, and $\sum_{i\in I_1} y_i = \sum_{i\in I_2} y_i = 0$.

Now, for $s = 1, 2$ let $\pi_s$ be a $p_{d-1}(m)$-uniform permutation on $I_s$ which is $c_{d-1}$-balanced with respect to $\{y_i\}_{i\in I_s}$. Call $\pi_1$ *$\alpha$-good* if for every $k_1$ with $1 \le k_1 \le m_1$ we have

$$\left|\sum_{i=1}^{k_1} w_{\pi_1(i)}^d - k_1\mu_1\right| \le 2\alpha\sqrt{k_1^*}, \tag{5.5}$$

where $k_1^* = \min\{k_1, m_1 - k_1\}$. In similar fashion to the proof of Lemma 6.1, Hoeffding's bounds [8] imply that for a particular value of $k_1$ we have

$$\Pr_{\text{unif}}[\pi_1 \text{ does not satisfy (5.5)}] \le 2\exp(-2\alpha^2),$$

and since the event depends only on the initial segment $\pi_1\{1, \ldots, k_1\}$, we also have

$$\begin{aligned}\Pr[\pi_1 \text{ does not satisfy (5.5)}] &\le p_{d-1}(m) \cdot \Pr_{\text{unif}}[\pi_1 \text{ does not satisfy (5.5)}] \\ &\le p_{d-1}(m) \cdot 2\exp(-2\alpha^2).\end{aligned}$$

Hence

$$\Pr[\pi_1 \text{ is not } \alpha\text{-good}] \le m p_{d-1}(m) \cdot 2\exp(-2\alpha^2). \tag{5.6}$$

Suppose that for some constants $C$ and $r$, the polynomial $p_{d-1}$ satisfies $p_{d-1}(k) \le Ck^r$ for all $k$. If we let $\alpha = \sqrt{(r+1)\ln(m)}$, then the right-hand side of (5.6) is at most $2Cm^{r+1-2(r+1)} \le \frac{1}{2}$ for sufficiently large $m$. Thus, we can assume that $\pi_1$ is $\alpha$-good with probability 1 and only increase $C$ by a constant factor. Similar arguments apply to $\pi_2$.

Finally, note that it is always possible to interleave $\pi_1$ and $\pi_2$ to give a permutation on $\{1, \ldots, m\}$ which is 1-balanced with respect to $\{w_i^d\}_{i=1}^m$. Let $\pi$ be such a permutation. Then we have $|\sum_{i=1}^k w_{\pi(i)}^d| \le 2$, and

$$\begin{aligned}\left|\sum_{i=1}^k w_{\pi(i)}^j\right| &= \left|\sum_{\substack{i\in I_1:\\ i\le k}} w_{\pi(i)}^j + \sum_{\substack{i\in I_2:\\ i\le k}} w_{\pi(i)}^j\right| \\ &= \left|\sum_{\substack{i\in I_1:\\ i\le k}} y_{\pi(i)}^j + \sum_{\substack{i\in I_2:\\ i\le k}} y_{\pi(i)}^j + \gamma^j \sum_{i=1}^k w_{\pi(i)}^d\right| \\ &\le \left|\sum_{\substack{i\in I_1:\\ i\le k}} y_{\pi(i)}^j\right| + \left|\sum_{\substack{i\in I_2:\\ i\le k}} y_{\pi(i)}^j\right| + |\gamma^j|\left|\sum_{i=1}^k w_{\pi(i)}^d\right| \\ &\le 4c_{d-1} + 4c_{d-1} + 2|\gamma^j| \\ &\le 8c_{d-1} + 2\end{aligned}$$

for all $j < d$ and $k$. Hence $\pi$ is $c_d'$-balanced for $c_d' = 4c_{d-1} + 1$ by assumption (5.3).

To verify that $\pi$ is almost uniform, we follow the proof of Lemma 3.3. Let $U \subseteq \{1, \ldots, m\}$ be arbitrary with $|U| = k$, and let $U_1 = U \cap I_1$, $U_2 = U \cap I_2$, $k_1 = |U_1|$, and $k_2 = |U_2|$. Then we have

$$\begin{aligned}\Pr\big[\pi\{1, \ldots, k\} = U\big] &\le \Pr\big[\pi_1\{1, \ldots, k_1\} = U_1 \text{ and } \pi_2\{1, \ldots, k_2\} = U_2\big] \\ &\le (Cm^r)^2 \Pr_{\text{unif}}\big[\pi_1\{1, \ldots, k_1\} = U_1 \text{ and } \pi_2\{1, \ldots, k_2\} = U_2\big] \\ &= \frac{C'm^{2r}}{\binom{m_1}{k_1}\binom{m_2}{k_2}}.\end{aligned}$$

Now we can bound the quantity $\binom{m_1}{k_1}\binom{m_2}{k_2}$ by mimicking (with minor modifications) the calculations from (3.7) to (3.10) in the proof of Lemma 3.3. In our current setting, we have $\mu_1, \mu_2 \ge \frac{1}{2d}$, and the $|w_i^d|$ are in $[0, 2]$. Because we have changed the definition of $\alpha$-good and the value of $\alpha$, we also have to make the substitutions $(B-1)^2 \to 2^2$ and $\ln m \to (r+1)\ln m$, respectively. Thus the bound on the exponent given in (3.9) becomes

$$\frac{10 \cdot 2^2}{(\frac{1}{2d})^2}(r+1)\ln m = 160d^2(r+1)\ln m. \tag{5.7}$$

Hence $\pi$ is $p_d(m)$-uniform for $p_d(m) = C'' m^{160d^2(r+1)+2r+1/2}$.

We have shown how to make balanced almost uniform permutations if the weights satisfy (5.3) and (5.4). To generalize to arbitrary weights $\{w_i\}_{i=1}^m$, we use the interval trick introduced in section 3.2. Let $\beta$ be a uniform random permutation in $\mathcal{S}_m$, and let $T_1 = \min\{t : |\sum_{i=1}^t w_i^j| > M^j \text{ for some } j\}$. Define $T_2, T_3, \ldots$ similarly. Now use the $T_i$ to divide $\beta$ into intervals $I_1, \ldots, I_q$. Let $\{\alpha_i\}_{i=1}^{q-1}$ be the aggregated ($d$-dimensional) weights of the first $q-1$ intervals. Note that if we divide each $\alpha_i^j$ by $\frac{1}{2}\max_i |\alpha_i^j|$, then the resulting weights satisfy (5.3) and (5.4). Hence these weights admit a $c'_d$-balanced, $p_d(q)$-uniform permution (though see the remark immediately following this proof). Rearranging the intervals $\{I_i\}_{i=1}^{q-1}$ according to such a permutation gives a permutation on $\{1, \ldots, m\}$ which is $p_d(m)$-uniform and $c_d$-balanced for $c_d = 2c'_d + 1$. ◻

*Remark.* We should point out that the weights $\{\alpha_i\}_{i=1}^{q-1}$ of the first $q-1$ intervals will not in general sum to zero. However, we can easily get around this by introducing a dummy weight $\alpha_q$ which is equal to the weight of interval $I_q$. The presence of this single small weight does not affect (5.7) for sufficiently large $m$. Hence there is a $c'_d$-balanced, $p_d$-uniform permutation on this padded sequence $\{\alpha_i\}_{i=1}^{q}$. This induces a permutation on $\{\alpha_i\}_{i=1}^{q-1}$ which is $(c'_d+1)$-balanced and $cp_d(q)$-uniform for some constant $c$. Thus, if we incorporate an extra $+1$ into the constant $c'_d$ and an extra factor of $c$ into $p_d$, then the argument in the above proof is still valid.

Before we specify our flow, we need one more definition.

DEFINITION 5.4. *Let $\{w_i\}_{i=1}^m$ be a sequence in $\mathbf{R}^d$, with $w_i = (w_i^1, \ldots, w_i^d)$, let $\mu = (\mu^1, \ldots, \mu^d) = \frac{1}{m}\sum_{i=1}^m w_i$, and let $l$ be a positive integer. A permutation $\pi$ is* strongly $l$-balanced *if, for all $k \le m$ and $j \le d$, there exists a set $S \subseteq \{1, \ldots, m\}$ with $|S \oplus \pi\{1, \ldots, k\}| \le l$ such that $(\sum_{i=1}^k w_{\pi(i)}^j - k\mu^j)$ and $(\sum_{i \in S} w_{\pi(i)}^j - k\mu^j)$ have opposite signs (or either is $0$).*

Thus, in a strongly balanced permutation, whenever the initial segment $\{\pi(i)\}_{i=1}^k$ is "above average" with respect to a particular coordinate $j$, it can be made "below average" by flipping at most some fixed number $l$ of items, and vice versa. As the name suggests, the strong balance condition is stricter than the usual balance condition. Nonetheless, the following lemma says that strongly balanced permutations always exist.

LEMMA 5.5. *For any sequence $\{w_i\}_{i=1}^m$ in $\mathbf{R}^d$, there exists a strongly $16d^2$-balanced permutation.*

Note that this lemma claims only that a *single* strongly balanced permutation exists; unlike Theorem 5.3, it makes no claims regarding almost uniformity. The proof of the lemma relies heavily on the result of Grinberg and Sevast'yanov quoted earlier (Lemma 5.2); the proof is straightforward but rather technical, so we defer it to section 6.

**5.3. A good flow.** Now that we have multidimensional balanced almost uniform permutations and strongly balanced permutations, we are ready to construct a good flow.

LEMMA 5.6. *Fix any number of knapsack constraints $d$. For arbitrary item weights, it is possible to construct a multicommodity flow $f$ in $G_\Omega$ with $\mathcal{C}(f)$ bounded by a polynomial in $n$ and $\mathcal{L}(f) = \mathrm{O}(n)$.*

*Proof.* Recall that we identify a vertex $\mathbf{x} \in \Omega$ with the index set $X = \{i : x_i = 1\}$. Let $\widehat{\Omega} = \{X \in \Omega : a^j(X) \le b^j - 3c_d \max_{i\in X} a_i^j\}$, where $c_d$ is the constant in the construction of balanced almost uniform permutations as in Theorem 5.3. Our main goal will be to construct a flow $\widehat{f}$ which, simultaneously for every $X, Y \in \widehat{\Omega}$, sends one unit of flow from $X$ to $Y$. This flow will satisfy $\mathcal{C}(\widehat{f}) \le \mathrm{poly}(n)$ and $\mathcal{L}(\widehat{f}) = \mathrm{O}(n)$.

Note that, from any vertex $X \in \Omega$, we can obtain a vertex $\widehat{X} \in \widehat{\Omega}$ by removing at most $3dc_d$ items. Thus, we can use an approach similar to that in section 4.1 to extend $\widehat{f}$ to a multicommodity flow $f$ on the whole of $\Omega$, and $f$ will satisfy $\mathcal{C}(f) \le \mathrm{O}(n^{6dc_d})\,\mathrm{poly}(n) \le \mathrm{poly}'(n)$ and $\mathcal{L}(f) \le \mathcal{L}(\widehat{f}) + 6dc_d = \mathrm{O}(n)$.

It remains to define the flow $\widehat{f}$ and show that it has the properties claimed. Fix $X, Y \in \widehat{\Omega}$. As in the one-dimensional case, the high-level idea is to route the flow from $X$ to $Y$ according to a balanced almost uniform permutation on the symmetric difference $X \oplus Y$. Recall from the one-dimensional case that we want both path endpoints to be far enough below the hyperplane before we apply the balanced permutation; recall also that we cannot in general ensure this by removing only a fixed number of items from $X$ and $Y$ (e.g., if the items in $X$ are much larger than those in $Y$). In the one-dimensional case (see section 4.2), we overcame this obstacle using the preprocessing random walk. Here we will use a different mechanism based on considering the "large" and the "small" items in $X \oplus Y$ separately. This mechanism is cruder but more robust and works in the multidimensional setting.

Let $M = (M^1, \dots, M^d)$, where $M^j = \min\{\max_{i\in X} a_i^j, \max_{i\in Y} a_i^j\}$. Let $L = \{i \in X \oplus Y : a_i^j > M^j \text{ for some } j\}$ and $S = (X \oplus Y) - L$. ($L$ and $S$ are the "large" and "small" items, respectively.) Let $\{w_i\}_{i\in X\oplus Y}$ be an enumeration of the weights of the items in $X \oplus Y$, where weights from $Y$ appear with a positive sign and weights from $X$ appear with a negative sign. Let $\mu_1 = \frac{1}{|L|}\sum_{i\in L} w_i$, and let $\mu_2 = \frac{1}{|S|}\sum_{i\in S} w_i$. Let $\pi_1$ be a permutation on $L$ which is strongly $16d^2$-balanced with respect to the weights $\{w_i\}_{i\in L}$, and let $\pi_2$ be a $p_d(|S|)$-uniform permutation which is $c_d$-balanced with respect to the weights $\{w_i - \mu_2\}_{i\in S}$. The existence of $\pi_1$ and $\pi_2$ is guaranteed by Lemma 5.5 and Theorem 5.3, respectively. To obtain $\pi$, we will interleave the strongly balanced permutation $\pi_1$ and the balanced permutation $\pi_2$. The rule for interleaving will be as follows. Suppose that $\pi(1), \dots, \pi(k)$ have already been assigned and that $\pi\{1, \dots, k\} = \pi_1\{1, \dots, k_1\} \cup \pi_2\{1, \dots, k_2\}$. Now, either $\frac{k_1}{k} \le \frac{|L|}{|L|+|S|}$ or $\frac{k_2}{k} < \frac{|S|}{|L|+|S|}$, so we can define $\pi(k+1)$ by

$$\pi(k+1) = \begin{cases} \pi_1(k_1+1) & \text{if } \frac{k_1}{k} \le \frac{|L|}{|L|+|S|}, \\ \pi_2(k_2+1) & \text{if } \frac{k_2}{k} < \frac{|S|}{|L|+|S|}. \end{cases}$$

Now let $\mu = \frac{1}{|X\oplus Y|}\sum_{i\in X\oplus Y} w_i = \frac{|L|\mu_1 + |S|\mu_2}{|L|+|S|}$. We claim that $\pi$ satisfies the following condition. Fix $j$ and $k$. Then there exist sets of indices $V_1$ and $V_2$, with $|V_i \oplus \{1, \dots, k\}| \le 17d^2$, such that

$$\sum_{i\in V_1} w_{\pi(i)}^j \le (k-1)\mu^j + 3c_d M^j, \tag{5.8}$$

$$\sum_{i\in V_2} w_{\pi(i)}^j \ge (k-1)\mu^j - 3c_d M^j. \tag{5.9}$$

We will prove this in the case ${\mu_1}^j \geq {\mu_2}^j$; if ${\mu_1}^j < {\mu_2}^j$, the proof is similar. Again, let $k_1 = |L \cap \pi\{1, \dots, k\}|$ and $k_2 = |S \cap \pi\{1, \dots, k\}|$, so that $\pi\{1, \dots, k\} = \pi_1\{1, \dots, k_1\} \cup \pi_2\{1, \dots, k_2\}$. The method of interleaving ensures that

$$\frac{k_1 - 1}{k - 1} \leq \frac{|L|}{|L| + |S|}, \quad \frac{k_2 - 1}{k - 1} \leq \frac{|S|}{|L| + |S|}.$$

Therefore, since ${\mu_1}^j \geq {\mu_2}^j$, we have

$$(k_1 - 1)\mu_1^j + k_2 \mu_2^j \leq (k - 1)\mu^j, \tag{5.10}$$

$$k_1 \mu_1^j + (k_2 - 1)\mu_2^j \geq (k - 1)\mu^j. \tag{5.11}$$

Clearly, the strong balance condition on $\pi_1$ implies that there exist $A, A'$, with $|A \oplus \{1, \dots, k_1\}| \leq 16d^2 + 1$ and similarly for $A'$, such that

$$\sum_{i \in A} w^j_{\pi_1(i)} \leq (k_1 - 1)\mu_1^j, \tag{5.12}$$

$$\sum_{i \in A'} w^j_{\pi_1(i)} \geq k_1 \mu_1^j. \tag{5.13}$$

Also, by the balance condition on $\pi_2$ we have

$$\sum_{i=1}^{k_2} w^j_{\pi_2(i)} \leq k_2 \mu_2^j + c_d \max_{i \in S}\{|w_i^j - \mu_2^j|\} \leq k_2 \mu_2^j + 3c_d M^j, \tag{5.14}$$

$$\sum_{i=1}^{k_2} w^j_{\pi_2(i)} \geq k_2 \mu_2^j - c_d \max_{i \in S}\{|w_i^j - \mu_2^j|\} \geq (k_2 - 1)\mu_2^j - 3c_d M^j. \tag{5.15}$$

Now, let $B = \pi^{-1}(\pi_1(A) \cup \pi_2\{1, \dots, k_2\})$ and $B' = \pi^{-1}(\pi_1(A') \cup \pi_2\{1, \dots, k_2\})$. Then we have $|B \oplus \{1, \dots, k\}| \leq 16d^2 + 1 \leq 17d^2$, and

$$\sum_{i \in B} w^j_{\pi(i)} = \sum_{i \in A} w^j_{\pi_1(i)} + \sum_{i=1}^{k_2} w^j_{\pi_2(i)}.$$

Exactly analogous relations hold with $B, A$ replaced by $B', A'$. Combining this with (5.10)–(5.15) gives (5.8) and (5.9).

Now, $\pi$ determines a path $\{Z_i\}_{i=0}^{|X \oplus Y|}$ from $X$ to $Y$, where $Z_0 = X$ and $Z_i = X \oplus \{\pi(1), \dots, \pi(i)\}$ for $1 \leq i \leq |X \oplus Y|$. This path might not stay in $\Omega$, but we can alter it slightly so that it does. Inequalities (5.8) and (5.9) imply that for every $k$ and $j$, there exists a set of indices $W_k^j$ with $|W_k^j| \leq 34d^2$ such that

$$a^j(Z_k - W_k^j) \leq \max\{a^j(X), a^j(Y)\} + 3c_d M^j \leq b^j, \tag{5.16}$$

$$a^j(Z_k \cup W_k^j) \geq \min\{a^j(X), a^j(Y)\} - 3c_d M^j. \tag{5.17}$$

Let $W_0 = \emptyset$ and for $k = 1, \dots, |X \oplus Y|$, let $W_k = \bigcup_{j=1}^d W_k^j$. Then, for all $k$, $|W_k| \leq 34d^3$ and $a(Z_k - W_k) \leq \mathbf{b}$. For $0 \leq k \leq |X \oplus Y|$, define

$$\overline{Z}_k = Z_k - W_k.$$

Then each $\overline{Z}_k \in \Omega$. Our flow from $X$ to $Y$ will pass through each of the $\overline{Z}_k$ in turn. To get from $\overline{Z}_k$ to $\overline{Z}_{k+1}$, we perform the following steps:

1. Remove each item in $\overline{Z}_k - (\overline{Z}_k \cap \overline{Z}_{k+1})$ in index order.
2. Add each item in $\overline{Z}_{k+1} - (\overline{Z}_k \cap \overline{Z}_{k+1})$ in index order.

Define $W_x = (W_k \cup W_{k+1}) \cap X$ and $W_y = (W_k \cup W_{k+1}) \cap Y$. By analogy with section 4, for each intermediate point $Z$ along the path define the "encoding" $\widetilde{Z}$ by

$$\widetilde{Z} = (X \oplus Y - Z) \cup (X \cap Y) - (W_k \cup W_{k+1}),$$

and let $U = \pi\{1, \dots, k\}$. By analogy with our earlier analysis, one can see that, for a given $Z$, $X$ and $Y$ are completely specified by the 4-tuple $(\widetilde{Z}, U, W_x, W_y)$. We also have

$$\begin{aligned} a^j(\widetilde{Z}) &= a^j(X) + a^j(Y) - a^j(Z \cup W_k \cup W_{k+1}) \\ &\le a^j(X) + a^j(Y) - \min\{a^j(Z_k \cup W_k), a^j(Z_{k+1} \cup W_{k+1})\} \\ &\le a^j(X) + a^j(Y) - (\min\{a^j(X), a^j(Y)\} - 3c_d M^j) \\ &= \max\{a^j(X), a^j(Y)\} + 3c_d M^j \\ &\le b^j, \end{aligned}$$

where the second inequality follows from (5.17). Hence $\widetilde{Z} \in \Omega$. We can therefore bound the flow $\widehat{f}(Z)$ through $Z$ by

$$\widehat{f}(Z) \le \sum_{\widetilde{Z} \in \Omega} \sum_{W_x, W_y, U} \Pr\big[\{\pi(1), \dots, \pi(|U|)\} = U\big]. \tag{5.18}$$

Finally, for a given $X$ and $Y$, let $L^j = \{i \in X \cup Y : a_i^j > M^j\}$, so that $L = \bigcup_{j=1}^d L^j$. Note that for every $j$, $L^j \cap Y$ is equal to either $L^j$ or $\emptyset$. Thus, if we define $k_2 = |U \cap S|$, then for given values of $M$ and $k_2$, there are at most $2^d\binom{|S|}{k_2}$ possible values for $U$ in the inner sum of (5.18). Therefore, we have

$$\begin{aligned} \widehat{f}(Z) &\le \sum_{\widetilde{Z} \in \Omega} \sum_{M, W_x, W_y, k_2} \sum_{U : |U \cap S| = k_2} \Pr\big[\{\pi_2(1), \dots, \pi_2(k_2)\} = U \cap S\big] \\ &\le \sum_{\widetilde{Z} \in \Omega} \sum_{M, W_x, W_y, k_2} \sum_{U : |U \cap S| = k_2} p_d(|S|) \binom{|S|}{k_2}^{-1} \\ &\le \sum_{\widetilde{Z} \in \Omega} \sum_{M, W_x, W_y, k_2} 2^d p_d(|S|) \\ &\le \sum_{\widetilde{Z} \in \Omega} n^d \left[\binom{n}{68d^3} 2^{68d^3}\right] n \times 2^d p_d(n) \\ &= \mathrm{poly}(n)|\Omega|, \end{aligned}$$

where in the second line we have appealed to the almost uniformity of permutation $\pi_2$. This completes the proof. ◻

Given such a flow, we can appeal to Theorem 2.1 to derive the main result of this section.

THEOREM 5.7. *Fix any dimension $d > 0$, and let $\Omega$ be the set of solutions to an arbitrary instance of the $d$-dimensional* 0-1 *knapsack problem. The mixing time of the random walk on $G_\Omega$ is* $\mathrm{poly}_d(n)$.

As in one dimension, this immediately implies the existence of an fpras for computing $|\Omega|$ in this more general setting.

*Remark.* In the multidimensional case we have made no attempt to optimize the exponent in our polynomial bound on the mixing time. Indeed, tracing through the proof of Lemma 5.6, we see that the bound is of the form $n^{\mathrm{O}((3d)!)}$. This is much larger than the lower bound $n^{\Omega(d)}$ mentioned earlier. It would be interesting to try to close this gap.

**6. Proofs of technical lemmas.** In this final section we supply the proofs of some technical results that were omitted from the main text.

**6.1. Lemmas for section 3.1.** The following two lemmas were used in the proof of Lemma 3.3.

LEMMA 6.1. *Let $\nu$ be a uniform random permutation in $\mathcal{S}_m$. Then*

$$\Pr[\nu \textit{ is not } \alpha\textit{-good}] \le 2m\exp(-2\alpha^2).$$

*Proof.* We adopt the notation of the proof of Lemma 3.3. Let $1 \le k_1 \le m_1$. It suffices to show that

$$\Pr\left[\left|\sum_{i=1}^{k_1} w_{\nu_1(i)} - k_1\mu_1\right| > \alpha(M-1)\sqrt{k_1}\right] \le 2\exp(-2\alpha^2),$$

for then the lemma follows from the union bound and from symmetry (which allows us to replace $k_1$ by $k_1^* = \min\{k_1, m_1 - k_1\}$). But this inequality is a direct consequence of Hoeffding's bound on deviations in sampling without replacement [8]. □

LEMMA 6.2. *Let $m_1, m_2, k_1, k_2$ be nonnegative integers and $m = m_1 + m_2$, $k = k_1 + k_2$. Then*

$$\frac{\binom{m_1}{k_1}\binom{m_2}{k_2}}{\binom{m}{k}} \ge Am^{-1/2}\exp\left\{-\frac{l^2 + \frac{1}{2}|l|}{\gamma(1-\gamma)}\left(\frac{1}{m_1} + \frac{1}{m_2}\right)\right\},$$

*where $\gamma = \frac{k}{m}$, $l = \frac{m_1k_2 - m_2k_1}{m}$, and $A > 0$ is a universal constant.*

*Proof.* Note that $k = \gamma m$, $k_1 = \gamma m_1 - l$, and $k_2 = \gamma m_2 + l$. By the symmetry of binomial coefficients, we may assume that $l \ge 0$. We shall prove the lemma by showing the two inequalities

$$\frac{\binom{m_1}{\gamma m_1}\binom{m_2}{\gamma m_2}}{\binom{m}{\gamma m}} \ge A_1 m^{-1/2} \tag{6.1}$$

and

$$\frac{\binom{m_1}{\gamma m_1 - l}\binom{m_2}{\gamma m_2 + l}}{\binom{m_1}{\gamma m_1}\binom{m_2}{\gamma m_2}} \ge A_2\exp\left\{-\frac{l^2 + \frac{1}{2}|l|}{\gamma(1-\gamma)}\left(\frac{1}{m_1} + \frac{1}{m_2}\right)\right\} \tag{6.2}$$

for positive constants $A_1, A_2$.

The first inequality is an immediate consequence of Stirling's approximation, $\sqrt{2\pi n}(\frac{n}{\mathrm{e}})^n \le n! \le A_3\sqrt{2\pi n}(\frac{n}{\mathrm{e}})^n$, where $A_3 = 1 + \mathrm{e}^{1/12}$ is a constant. To prove the second inequality, we apply Stirling's approximation to all four binomial coefficients to get the following lower bound on the left-hand side of (6.2):

$$\begin{aligned}&\left[\frac{\mathrm{P}(\gamma m_1)\mathrm{P}((1-\gamma)m_1)\mathrm{P}(\gamma m_2)\mathrm{P}((1-\gamma)m_2)}{\mathrm{P}(\gamma m_1 - l)\mathrm{P}((1-\gamma)m_1 + l)\mathrm{P}(\gamma m_2 + l)\mathrm{P}((1-\gamma)m_2 - l)}\right] \\ &\times\left[\frac{\gamma m_1(1-\gamma)m_1\gamma m_2(1-\gamma)m_2}{(\gamma m_1 - l)((1-\gamma)m_1 + l)(\gamma m_2 + l)((1-\gamma)m_2 - l)}\right]^{1/2},\end{aligned} \tag{6.3}$$

where $\mathrm{P}(x)$ denotes $x^x$. Now we have

$$\frac{\mathrm{P}(\gamma m_1)}{\mathrm{P}(\gamma m_1 - l)} = (\gamma m_1)^l \left(1 + \frac{l}{\gamma m_1 - l}\right)^{\gamma m_1 - l}$$
$$\geq (\gamma m_1)^l \exp\left\{\frac{l(\gamma m_1 - l)}{\gamma m_1}\right\},$$

where we have used the inequality $(1 + \frac{x}{y})^y \geq \exp(\frac{xy}{x+y})$, valid for $x, y > 0$. Handling the three other pairs of factors in the numerator and denominator in a similar fashion (using in addition the inequality $(1 - \frac{x}{y})^y \geq \exp(\frac{-xy}{y-x})$, valid for $y > x > 0$), we see that the first parenthesis in (6.3) is bounded below by

$$\exp\left\{-\frac{l^2}{\gamma(1-\gamma)}\left(\frac{1}{m_1} + \frac{1}{m_2}\right)\right\}. \tag{6.4}$$

A similar calculation bounds the second parenthesis in (6.3) by

$$\exp\left\{-\frac{|l|}{\gamma(1-\gamma)}\left(\frac{1}{m_1} + \frac{1}{m_2}\right)\right\}. \tag{6.5}$$

Combining (6.4) and (6.5) completes the verification of inequality (6.2) above and hence the proof of the lemma. ☐

**6.2. Proof of Lemma 4.1.** By removing $X \cap Y$ from both $X$ and $Y$ and replacing $b$ by $b - a(X \cap Y)$, we may assume that $X \cap Y = \emptyset$. Moreover, by scaling all the $a_i$ and $b$ we may assume that $M = \max_i a_i = 1$. Finally, we may assume that $b \geq 3\Delta$ since otherwise there are no pairs $(X, Y)$ satisfying the hypothesis of the lemma.

Define

$$\mathcal{F} = \{(X', Y') : a(X') \geq b - \Delta \text{ or } a(Y') \geq b - \Delta\},$$
$$\mathcal{E} = \{(X', Y') : a(X') \leq b - 2\Delta \text{ and } a(Y') \leq b - 2\Delta\}.$$

Note that $\mathcal{F}$ contains all full pairs $(X', Y')$, and $\mathcal{E}, \mathcal{F}$ are disjoint. Also, define the hitting times

$$\mathcal{T} = \max_{(X',Y') \in \mathcal{F}} \mathrm{E}(\text{number of PRW steps to hit } \mathcal{E} \text{ starting at } (X', Y')),$$
$$\mathcal{U} = \min_{(X',Y') \in \mathcal{E}} \mathrm{E}(\text{number of PRW steps to hit } \mathcal{F} \text{ starting at } (X', Y')).$$

Now we claim that the lemma will follow if we can show the following:

(i) $\mathcal{T} \leq \alpha m$ for some constant $\alpha > 0$;

(ii) $\mathcal{U}/\mathcal{T} \geq \beta$ for some constant $\beta > 0$.

To see this, set the length of the PRW to be $C_1 m = 4\alpha m$, and let $(X_t, Y_t)$ denote the sequence of pairs visited by the PRW, with $(X_0, Y_0) = (X, Y) \in \mathcal{F}$. Let $T_0$ be the first time $t$ at which $(X_t, Y_t) \in \mathcal{E}$ (or $T_0 = C_1 m$ if the walk ends before this occurs); then let $U_1$ be the first $t$ for which $(X_{T_0+t}, Y_{T_0+t}) \in \mathcal{F}$, and $T_1$ the first $t$ for which $(X_{T_0+U_1+t}, Y_{T_0+U_1+t}) \in \mathcal{E}$. Continue defining a sequence of hitting times $U_2, T_2, U_3, T_3, \ldots$ in this way until the end of the walk is reached. Note that the PRW is not full for at least $\sum_i U_i$ steps and that $\sum_{i \geq 0} T_i + \sum_{i \geq 1} U_i = 4\alpha m$ is the total walk length. Now from facts (i) and (ii) we have

$$\mathrm{E}\left(\sum_{i \geq 0} T_i - \tfrac{1}{\beta} \sum_{i \geq 1} U_i\right) = \mathrm{E}\left(T_0 + \sum_{i \geq 1} (T_i - \tfrac{1}{\beta} U_i)\right) \leq \alpha m.$$

An application of Markov's inequality then ensures that $\sum_{i\geq 0} T_i - \frac{1}{\beta}\sum_{i\geq 1} U_i \leq 2\alpha m$ with probability at least $\frac{1}{2}$. Conditioning on this event, we have $(1+\frac{1}{\beta})\sum U_i \geq 2\alpha m$, and thus the proportion of steps during which the PRW is not full is at least $1/2(1+\frac{1}{\beta})$, a constant. The lemma now follows easily.

It remains to verify facts (i) and (ii) above: these are immediate consequences of the following two claims. Let $\sigma^2 = \frac{1}{m}\sum_{i\in X\cup Y} a_i^2$ be the second moment of the item weights, and note that $\sigma^2 \geq 1/m$ since $\max_i a_i = 1$.

Claim 1. *$\mathcal{T} \leq \gamma_1/\sigma^2$ for a constant $\gamma_1 > 0$.*

Claim 2. *$\mathcal{U} \geq \gamma_2/\sigma^2$ for a constant $\gamma_2 > 0$.*

*Proof of Claim* 1. Choose an initial pair $(X_0, Y_0) \in \mathcal{F}$ that maximizes the expected time until the PRW hits $\mathcal{E}$, and let $(X_t, Y_t)$ denote the PRW starting at $(X_0, Y_0)$. We may assume without loss of generality that $a(X_0) > a(Y_0)$, so that $a(X_0) \in [b-\Delta, b]$ and $\mathcal{T}$ is the expected time until $a(X_t) \leq b-2\Delta$. We estimate $\mathcal{T}$ by coupling the PRW with the *unconstrained* random walk, which behaves exactly like the PRW except that the constraint $\sum a_i \leq b$ is ignored. (Thus it is just simple random walk on an $m$-dimensional hypercube with holding probability $\frac{1}{2}$ at every step.) Write $(\widehat{X}_t, \widehat{Y}_t)$ for the unconstrained random walk, with $(\widehat{X}_0, \widehat{Y}_0) = (X_0, Y_0)$, and consider the first time $t = \widehat{T}$ at which $|a(\widehat{X}_t) - a(X_0)| \geq 2\Delta$. Now $a(\widehat{X}_t)$ is a supermartingale up to time $\widehat{T}$, since $\mathrm{E}(a(\widehat{X}_{t+1}) - a(\widehat{X}_t)|\widehat{X}_t) = \frac{1}{2m}(a(\widehat{Y}_t) - a(\widehat{X}_t)) < 0$. Thus with constant probability $a(\widehat{X}_{\widehat{T}}) \leq a(\widehat{X}_0) - 2\Delta$, and so $(X_{\widehat{T}}, Y_{\widehat{T}}) \in \mathcal{E}$. Hence $\mathcal{T}$ is bounded above by a constant times $\mathrm{E}(\widehat{T})$. But we also have $\mathrm{E}((a(\widehat{X}_{t+1}) - a(\widehat{X}_t))^2|\widehat{X}_t) = \frac{1}{2m}\sum_i a_i^2 = \frac{\sigma^2}{2}$. So $\mathrm{E}(\widehat{T})$ is the expected time for a supermartingale with increments bounded by $\pm 1$ and with second moment $\sigma^2/2$ to move a distance $\pm 2\Delta$ from its initial value. A standard application of the martingale optional stopping theorem (see, e.g., [6, section 12.5]) now yields that $\mathrm{E}(\widehat{T}) \leq \frac{(4\Delta+1)^2}{\sigma^2/2} = \gamma_1/\sigma^2$ for a positive constant $\gamma_1$. This completes the proof of Claim 1.

*Proof of Claim* 2. As above, let $(X_t, Y_t)$ denote the PRW, but now with $(X_0, Y_0) \in \mathcal{E}$. We follow the random variable $Z_t = a(X_t) - a(Y_t)$, which always has a drift toward 0 (i.e., $\mathrm{E}(Z_{t+1} - Z_t|X_t) \times Z_t \leq 0$ for all $t$). Note that initially $|Z_0| \leq 2(b-2\Delta) - V$, where $V = a(X_t) + a(Y_t) = \sum_i a_i$ is independent of $t$. And when $(X_t, Y_t) \in \mathcal{F}$ we have $|Z_t| \geq 2(b-\Delta) - V$. Thus $\mathcal{U}$ is bounded below by the minimum expected time for $|Z_t|$ to increase by $2\Delta$ from its initial value. But the second moment is $\mathrm{E}((Z_{t+1} - Z_t)^2|X_t) = \frac{1}{2m}\sum_i (2a_i)^2 = 2\sigma^2$, so by a similar application of the optional stopping theorem we conclude that $\mathcal{U} \geq \gamma_2/\sigma^2$, as claimed.

This completes the verification of Claims 1 and 2 and hence the proof of the lemma. □

**6.3. Proof of Lemma 5.5.** First suppose that $\sum_{i=1}^m w_i = 0$. We will show that, in this case, there exists a strongly $8d^2$-balanced permutation $\pi$. Let $L$ be the set containing the $4d$ indices $i$ with the largest values of $w_i^j$ and the $4d$ indices $i$ with the largest values of $-w_i^j$ for each $j \leq d$. Then $|L| \leq 8d^2$.

The permutation $\pi$ we construct will satisfy $\{\pi(m), \ldots, \pi(m-|L|+1)\} = L$. It will be enough to check that the strong balance condition holds for $1 \leq k \leq m-|L|$. It suffices to show that, for all $j \leq d$ and $k \leq m-|L|$, we have

$$-s^{j+} \leq \sum_{i=1}^{k} w_{\pi(i)}^j \leq s^{j-}, \tag{6.6}$$

where

$$s^{j+} \equiv \sum_{i\in L}(w_i^j)^+, \qquad s^{j-} \equiv \sum_{i\in L}(w_i^j)^-.$$

We will need the Grinberg–Sevast'yanov result (Lemma 5.2), which states that for any set of vectors $x_1,\dots,x_n$ in $\mathbf{R}^d$ with $\sum_i x_i = 0$, there exists a permutation $\nu \in \mathcal{S}_n$ such that

$$\sum_{i=1}^{k} x_{\nu(i)} \in d \times \operatorname{conv}\{x_1,\dots,x_n\} \quad \text{for } 1 \le k \le n.$$

Note that the permutation $\nu'$ defined by $\nu'(i) = \nu(n+1-i)$ for all $i$ satisfies

$$\sum_{i=1}^{k} x_{\nu'(i)} \in -d \times \operatorname{conv}\{x_1,\dots,x_n\} \quad \text{for } 1 \le k \le n,$$

because $\sum_{i=1}^{k} x_{\nu'(i)} = \sum_{i=m-k+1}^{m} x_{\nu(i)} = -\sum_{i=1}^{k} x_{\nu(i)}$.

Let $S_1 = \{1,\dots,m\} - L$, let $m' = m - |L|$, and let $\pi_1$ be a permutation on $S_1$ such that

$$\sum_{i=1}^{k}(w_{\pi_1(i)} - \mu_1) \in -d \times \operatorname{conv}\{w_i - \mu_1 : i \in S_1\}$$

for all $k$, where $\mu_1 = \frac{1}{m'}\sum_{i\in S_1} w_i$. Suppose that $m'$ is even and $m' = 2r$; if $m'$ is odd, the proof is similar. Now, let $S_2 = \{\pi_1(r+1),\dots,\pi_1(m')\}$, and let $\pi_2$ be a permutation on $S_2$ such that

$$\sum_{i=1}^{k}(w_{\pi_2(i)} - \mu_2) \in d \times \operatorname{conv}\{w_i - \mu_2 : i \in S_2\}, \tag{6.7}$$

where $\mu_2 = \frac{1}{r}\sum_{i\in S_2} w_i$. Define the permutation $\pi$ by

$$\pi(i) = \begin{cases} \pi_1(i) & \text{if } i \le r, \\ \pi_2(i-r) & \text{if } r < i \le m'. \end{cases}$$

We must check that $\pi$ satisfies (6.6). Fix $j$. Without loss of generality, $s^{j+} \le s^{j-}$, so that $\mu_1^j \ge 0$. For $k \le r$ we have

$$\begin{aligned}
\sum_{i=1}^{k} w^j_{\pi(i)} &= \sum_{i=1}^{k} w^j_{\pi_1(i)} \\
&\ge \sum_{i=1}^{k}(w^j_{\pi_1(i)} - \mu_1^j) \\
&\ge -d \max_{1\le i\le m'}\{w_i^j - \mu_1^j\} \\
&\ge -d \max_{1\le i\le m'}\{w_i^j\} \\
&\ge -\tfrac{1}{4}s^{j+} \qquad (6.8)\\
&\ge -s^{j+}
\end{aligned}$$

and

$$\begin{aligned}
\sum_{i=1}^{k} w^j_{\pi(i)} &= \sum_{i=1}^{k} w^j_{\pi_1(i)} \\
&= k\mu_1^j + \sum_{i=1}^{k}(w_{\pi_1(i)} - \mu_1^j) \\
&\le r\mu_1^j + d \max_{1\le i\le m'}\{-(w_i^j - \mu_1^j)\} \\
&= \tfrac{1}{2}(s^{j-} - s^{j+}) + d \max_{1\le i\le m'}\{-w_i^j\} + d\mu_1^j \\
&\le \tfrac{1}{2}(s^{j-} - s^{j+}) + \tfrac{1}{4}s^{j-} + \tfrac{1}{4}s^{j+} \\
&= \tfrac{3}{4}s^{j-} - \tfrac{1}{4}s^{j+} \\
&\le s^{j-} - \tfrac{1}{4}s^{j+}. \qquad (6.9)
\end{aligned}$$

(We will need the extra $-\frac{1}{4}s^{j+}$ in the second part of the proof.) For $r < k \le 2r$ we have

$$\sum_{i=1}^{k} w^j_{\pi(i)} = \sum_{i=1}^{r} w^j_{\pi_1(i)} + \sum_{i=1}^{k-r} w^j_{\pi_2(i)}. \qquad (6.10)$$

Now if $\mu_2^j < 0$, then the conditional expectation of $\sum_{i=1}^{k} w^j_{\pi(i)}$ given $\pi_1$ is at least $-s^{j+} + s^{j-}$. Hence we must have

$$\begin{aligned}
\sum_{i=1}^{k} w^j_{\pi(i)} &\ge -s^{j+} + s^{j-} - d \max_{i\in S_2}\{-(w_i^j - \mu_2^j)\} \\
&\ge -s^{j+} + s^{j-} - d \max_{i\in S_2}\{-w_i^j\} \\
&\ge -s^{j+} + s^{j-} - \tfrac{1}{4}s^{j-} \\
&\ge -s^{j+}.
\end{aligned}$$

On the other hand, if $\mu_2^j \ge 0$, the right-hand side of (6.10) can be bounded below as follows:

$$\begin{aligned}
\sum_{i=1}^{r} w^j_{\pi_1(i)} + \sum_{i=1}^{k-r} w^j_{\pi_2(i)} &= \left[\sum_{i=1}^{r} w^j_{\pi_1(i)} + (k-r)\mu_2^j\right] + \sum_{i=1}^{k-r}(w^j_{\pi_2(i)} - \mu_2^j) \\
&\ge \left[\tfrac{1}{2}(s^{j-} - s^{j+}) - d \max_{1\le i\le m'}\{w_i^j - \mu_1^j\}\right] - d \max_{i\in S_2}\{-(w_i^j - \mu_2^j)\} \\
&\ge \left[\tfrac{1}{2}(s^{j-} - s^{j+}) - d \max_{1\le i\le m'}\{w_i^j\}\right] - d \max_{i\in S_2}\{-w_i^j\} - d\mu_2^j \\
&\ge \left[\tfrac{1}{2}(s^{j-} - s^{j+}) - \tfrac{1}{4}s^{j+}\right] - \tfrac{1}{4}s^{j-} - \tfrac{1}{4}s^{j-} \\
&= -\tfrac{3}{4}s^{j+} \\
&\ge -s^{j+}.
\end{aligned}$$

For a corresponding upper bound, we can write

$$\sum_{i=1}^{k} w^j_{\pi(i)} \le \sum_{i=1}^{r} w^j_{\pi_1(i)} + (k-r)\mu_2^j + \min\{d, k-r\} \max_{i\in S_2}\{w_i^j - \mu_2^j\}. \qquad (6.11)$$

If $\mu_2^j \geq 0$, the right-hand side of (6.11) is bounded above by

$$(s^{j-} - s^{j+}) + d \max_{i \in S_2}\{w_i^j\} \leq (s^{j-} - s^{j+}) + \tfrac{1}{4}s^{j+} \leq s^{j-}.$$

If $\mu_2^j < 0$, the right-hand side of (6.11) is bounded above by

$$\sum_{i=1}^{r} w_{\pi_1(i)}^j + \min\{d, k-r\} \max_{i \in S_2}\{w_i^j\} \leq (s^{j-} - \tfrac{1}{4}s^{j+}) + \tfrac{1}{4}s^{j+} \leq s^{j-},$$

where in the first inequality we have used (6.9).

Putting all the above together, we see that $\pi$ is strongly $8d^2$-balanced. Furthermore, in light of (6.6), we know also that, for all $k$ and $j$, there exists a set of indices $S \supseteq \{1, \ldots, k\}$, with $|S| \leq k+8d^2$, such that $\sum_{i=1}^{k} w_{\pi(i)}^j$ and $\sum_{i \in S} w_{\pi(i)}^j$ have opposite signs.

Now let $\{w_i\}_{i=1}^m$ be arbitrary. From the above, there exists a permutation $\pi$ which is strongly $8d^2$-balanced with respect to the sequence $\{w_i - \mu\}_{i=1}^m$, where $\mu = \frac{1}{m}\sum_{i=1}^m w_i$. We claim that $\pi$ is also strongly $16d^2$-balanced with respect to the original sequence $\{w_i\}_{i=1}^m$. Fix $k$ and $j$, and define $\mu^j = \frac{1}{m}\sum_{i=1}^m w_i^j$. Without loss of generality, $\mu^j \geq 0$. Then there exists $S \supseteq \{1, \ldots, k\}$ with $|S| \leq k + 8d^2$ such that

$$\sum_{i \in S} w_{\pi(i)}^j - k\mu^j \geq \sum_{i \in S} w_{\pi(i)}^j - |S|\mu^j \geq 0.$$

In addition, there exists $S \supseteq \{1, \ldots, k\}$, with $|S| \leq k + 8d^2$ such that $\sum_{i \in S} w_{\pi(i)}^j \leq |S|\mu^j$. It follows that, for some $S' \subseteq S$ with $|S'| = k$, we have $\sum_{i \in S'} w_{\pi(i)}^j \leq k\mu^j$. Since $|S' \oplus \{1, \ldots, k\}| \leq 16d^2$, this completes the proof. ☐

**Acknowledgment.** We thank Persi Diaconis for bringing reference [7] to our attention.

## REFERENCES

[1] M. Cryan, M. Dyer, L. Goldberg, M. Jerrum, and R. Martin, *Rapidly mixing Markov chains for sampling contingency tables with a constant number of rows*, in Proceedings of the 43rd IEEE Symposium on Foundations of Computer Science, IEEE Computer Society, Los Alamitos, CA, 2002, pp. 711–720.

[2] P. Dagum, M. Luby, M. Mihail, and U. Vazirani, *Polytopes, permanents and graphs with large factors*, in Proceedings of the 29th IEEE Symposium on Foundations of Computer Science, IEEE Computer Society, Los Alamitos, CA, 1988, pp. 412–421.

[3] P. Diaconis and D. Stroock, *Geometric bounds for eigenvalues of Markov chains*, Ann. Appl. Probab., 1 (1991), pp. 36–61.

[4] M. Dyer, *Approximate counting by dynamic programming*, in Proceedings of the 35th ACM Symposium on Theory of Computing, ACM, New York, 2003, pp. 693–699.

[5] M. Dyer, A. Frieze, R. Kannan, A. Kapoor, L. Perkovic, and U. Vazirani, *A mildly exponential time algorithm for approximating the number of solutions to a multidimensional knapsack problem*, Combin. Probab. Comput., 2 (1993), pp. 271–284.

[6] G. R. Grimmett and D. R. Stirzaker, *Probability and Random Processes*, 3rd ed., Oxford University Press, Oxford, UK, 1992.

[7] V. Grinberg and S. Sevast'yanov, *Value of the Steinitz constant*, Funktsional. Anal. i Prilozhen., 14 (1980), pp. 56–57.

[8] W. Hoeffding, *Probability inequalities for sums of bounded random variables*, J. Amer. Statist. Assoc., 58 (1963), pp. 13–30.

[9] M. R. Jerrum, *Mathematical foundations of the Markov chain Monte Carlo method*, in Probabilistic Methods for Algorithmic Discrete Mathematics, M. Habib, C. McDiarmid, J. Ramirez-Alfonsin, and B. Reed, eds., Algorithms Combin. 16, Springer-Verlag, Berlin, 1998, pp. 116–165.

[10] M. R. Jerrum and A. J. Sinclair, *Approximating the permanent*, SIAM J. Comput., 18 (1989), pp. 1149–1178.

[11] M. R. Jerrum and A. J. Sinclair, *Polynomial-time approximation algorithms for the Ising model*, SIAM J. Comput., 22 (1993), pp. 1087–1116.

[12] M. R. Jerrum and A. J. Sinclair, *The Markov chain Monte Carlo method: An approach to approximate counting and integration*, in Approximation Algorithms for NP-Hard Problems, D. S. Hochbaum, ed., PWS Publishing, Boston, 1997, pp. 482–520.

[13] M. R. Jerrum, L. G. Valiant, and V. V. Vazirani, *Random generation of combinatorial structures from a uniform distribution*, Theoret. Comput. Sci., 43 (1986), pp. 169–188.

[14] R. Kannan, *Markov chains and polynomial time algorithms*, in Proceedings of the 35th IEEE Conference on Foundations of Computer Science, IEEE Computer Society, Los Alamitos, CA, 1994, pp. 656–671.

[15] B. Morris, *Random Walks in Convex Sets*, Ph.D. thesis, University of California at Berkeley, Berkeley, CA, 2000.

[16] B. Morris and A. J. Sinclair, *Random walks on truncated cubes and sampling* 0-1 *knapsack solutions*, in Proceedings of the 40th IEEE Symposium on Foundations of Computer Science, IEEE Computer Society, Los Alamitos, CA, 1999, pp. 230–240.

[17] L. Saloff-Coste, *Lectures on finite Markov chains*, in Lectures on Probability Theory and Statistics, Lecture Notes in Math. 1665, Springer-Verlag, Berlin, 1997, pp. 301–413.

[18] A. J. Sinclair, *Improved bounds for mixing rates of Markov chains and multicommodity flow*, Combin. Probab. Comput., 1 (1992), pp. 351–370.

[19] A. J. Sinclair, *Algorithms for Random Generation and Counting*, Progr. Theoret. Comput. Sci., Birkhäuser Boston, Boston, MA, 1993.

SIAM J. COMPUT. © 2004 Society for Industrial and Applied Mathematics
Vol. 34, No. 1, pp. 227–247

# BALANCED-REPLICATION ALGORITHMS FOR DISTRIBUTION TREES[*]

EDITH COHEN[†] AND HAIM KAPLAN[‡]

**Abstract.** In many Internet applications, requests for a certain object are routed bottom-up over a tree where the root of the tree is the node containing the object. When an object becomes popular, the root node of the tree may become a hot-spot. Therefore, many applications allow intermediate nodes to acquire the ability to serve the requests, for example, by caching the object. We call such distinguished nodes *primed*. We propose and analyze different algorithms where nodes decide when to become primed; these algorithms balance the maximum load on a node and the number of primed nodes.

Many applications require both fully distributed decisions and smooth convergence to a stable set of primed nodes. We first present optimal algorithms which require communication across the tree. We then consider the natural previously proposed THRESHOLD algorithm, where a node becomes primed when the incoming flow of requests exceeds a threshold. We show examples where THRESHOLD exhibits undesirable behavior during convergence. Finally, we propose another fully distributed algorithm, GAP, which converges gracefully.

**Key words.** replication, peer-to-peer, cache, networks, distributed algorithms, algorithms

**AMS subject classifications.** 68W15, 68W20, 68W40, 68W25, 68M10, 68M12

**DOI.** 10.1137/S0097539702412143

## 1. Introduction.

The Internet provides a platform for decentralized applications where content or services are requested, stored, and provided by a very large number of loosely coupled hosts. In these applications, there is no centralized support. Requests are forwarded from peer to peer, and each peer can become a server for each item.

Many supporting architectures for such applications are such that the search pattern for each request is "routed"; that is, every peer has a list of neighbors, and for each request a peer might receive, there is a preferred neighbor that brings the request "closer" to its destination. Therefore, the forwarding pattern of each request forms a tree, with edges directed toward the root. We call this tree a *distribution tree.* The stream of requests traveling up the distribution tree is called a *flow.*

When a certain request becomes popular, the respective root of its distribution tree can become a hot-spot. In this case, hosts further down the tree can "replicate" the service, that is, acquire the ability to satisfy these requests, e.g., by "caching" data or duplicating code. The focus of this paper is proposing and evaluating algorithms for hosts to decide when to become replicas.

Before providing further details on our model, we describe some concrete applications and architectures where this setup arises. A relatively old existing application is the domain name service (DNS). The DNS already offers caching of records at intermediate nodes; in fact, caching is the default at all "intermediate" servers, and DNS records are cached until they expire. A forward-looking application is to place

[*]Received by the editors July 29, 2002; accepted for publication (in revised form) March 27, 2004; published electronically October 28, 2004. A preliminary version of this paper appeared in the Proceedings of the 10th European Symposium on Algorithms (ESA'2002) [4].
http://www.siam.org/journals/sicomp/34-1/41214.html

[†]AT&T Labs–Research, Florham Park, NJ 07932 (edith@research.att.com).

[‡]School of Computer Science, Tel-Aviv University, Tel-Aviv 69978, Israel (haimk@cs.tau.ac.il). This author's work was supported, in part, by the Israel Science Foundation (ISF) grant 548/00.

HTTP caches "near" Internet routers, augmenting these routers with the capability to intercept and forward HTTP traffic; such a service can shorten latency and network load. The growing popularity of peer-to-peer applications (e.g., [8, 2]) sparked the development of architectures for fully distributed name-lookup, which enable clients to locate objects or services. Two such proposed architectures are Chord [18] (which is based on consistent hashing [9]) and CAN (content addressable network) [15]. With Chord and CAN, objects have keys which are hashed to points in some metric space $S$. Each node is responsible for some segment of the space $S$. Note that the metric of $S$ has no correspondence to the underlying network distances. Each node maintains a small number of pointers $P(v)$ to nodes responsible for other regions of $S$. When a node receives a request for an object with key $k$, it forwards it to the node in $P(v)$ responsible for a region containing the point closest to the hash of $k$ (according to the metric on $S$). Thus, the search paths for each object form a distribution tree, with the root of the tree being the node responsible for the region of the metric space where the hash of the object lies. Note that even though these hashing-based architectures balance the load well across different objects, replication is still necessary for avoiding hot-spots caused by highly skewed per-object demands. Thus, to alleviate hot-spots, it is proposed in [15, 18] that intermediate nodes cache items (or keys).

In all these applications, replication on intermediate nodes constitutes a tool in avoiding "hot-spots" at the roots of distribution trees. Such hot-spots are caused by flash-crowd events or objects with persistent high demand. We would like to emphasize that generally, each distribution tree corresponds to a single "item," and thus one might worry about interaction of flows from different trees. In many of these applications, however, the balance between trees is taken care of by the process selecting these trees; e.g., in CAN and Chord, the trees are such that nodes are essentially assigned randomly to their positions in different trees. Thus, it is reasonable to consider the trees independently of each other and focus on a single tree, which is the problem we model and address in this paper.

We say that we *prime* a node when we give it the ability to serve requests. We refer to a set of primed nodes as an *allocation*. We look for algorithms that select an allocation. We model the problem and propose metrics for the quality of a solution. The objective is to balance two parameters: the maximum load of a node, which we would like to be as small as possible, and the total size of the allocation (number of primed nodes), which we would also like to be as small as possible. The objective of minimizing the number of primed nodes models a wide range of applications where being primed consumes resources (e.g., storage space). Other desirable properties pertain to the complexity and communication requirements of the algorithm and the stability of the allocation. We consider two models: The first, called the *must-serve* model, requires that a primed node will serve all the flow it gets. The second, called the *serve-or-forward* model, allows a node to serve some of the flow it gets and forward the rest to its parent.

In the first part of the paper we assume that there is a strict upper bound of $C$ on the amount of flow that a primed node can serve and on the amount of flow that a single leaf can generate. Then it is obvious that to serve a total of $R$ flow we need to prime at least $\lceil R/C \rceil$ nodes. We prove in section 3 that if the tree has large degrees, then we may in fact have to prime $\Omega(R\Delta/C)$ nodes, where $\Delta$ is the maximum degree.

In section 4 we consider the problem of finding the optimal allocation. Both for the *must-serve* and the *serve-or-forward* models, we present algorithms that given the tree and the flow generated at each leaf calculate a smallest allocation in time almost linear in the size of the tree. These algorithms can be implemented distributively but with some overhead of two rounds of passing messages up and down the tree.

In section 5 we consider a simple and natural algorithm proposed with both Chord and CAN which we call the THRESHOLD algorithm. The THRESHOLD algorithm runs locally at each node and primes the node when the incoming flow is above a certain threshold. An advantage of the THRESHOLD algorithm is that it requires no communication other than the incoming flow itself. We provide bounds on the size of the allocation obtained by THRESHOLD and show that it is no more than $R\Delta/C$, the same as the worst-case lower bound.

According to the THRESHOLD algorithm, the decision of a node whether to prime itself or not depends on its incoming flow, which in turn depends on the priming decisions of its descendants. This implies that when nodes run THRESHOLD asynchronously, they may change their state from primed to unprimed multiple times before reaching a stable allocation. This dependency may result in instability. In particular, when the incoming flow changes, the size of interim allocations may be much larger than the size of the final stable one. Furthermore, small changes in the incoming flow may cause large changes in the allocation. Such instability is a disadvantage for any application where there is a cost associated with a change of state from primed to unprimed or vice versa. For example, to serve requests for a particular big file you have to store a copy of the file. This may be an overhead even when this storage is only temporary, particularly if the node participates in several such trees.

Finally, in section 6, we propose a different local algorithm called GAP. Like THRESHOLD, GAP also runs independently at each node and makes a local decision. Unlike THRESHOLD, GAP makes its decision at a node $v$ based on the total flow generated at leaf descendants of $v$. Notice that the value of the total flow generated below a particular node is independent of the decisions made by other nodes. Thus GAP converges immediately to an allocation which is stable. Furthermore, GAP's allocation is unlikely to change significantly due to minor changes in the flow.

The algorithm GAP is randomized, and as a result a node primed by GAP may end up serving more than $C$ units of flow. We show that, with an appropriate choice of parameters, the probability that a primed node serves more than $C$ units of flow is small. Furthermore, we show that by running GAP iteratively logarithmically many times, one can find an allocation of expected size $O(\frac{R\Delta}{C})$ in which no node serves more than $C$ units of flow with high probability.

**Related work.** There has been extensive work on caching and replicating content in order to improve response time and reduce the load on particular servers. In particular, several groups have proposed and analyzed many aspects of cooperating caches [5, 7, 13, 3, 10]. This work is related to ours only in the sense that load is distributed among many caches. The main question considered in these papers is how to distribute the load among the cooperating caches assuming typically that each client can send requests to any cache. We ask a different question: Which nodes should we distinguish as caches so that there are no hot-spots on one hand, and not too many nodes become caches on the other hand?

A recent analysis of traffic of Internet content delivery systems [16] suggests that peer-to-peer traffic now accounts for the majority of HTTP bytes transferred, exceeding traffic due to WWW accesses by nearly a factor of three. Peer-to-peer documents are large, so peer-to-peer flows last longer, and many of them are concurrently open. All this, and the fact that a small number of extremely large objects accounts for a large fraction of the traffic, indicates the large possible gain of replication in a peer-to-peer system. The large file sizes also indicate that there is a nonnegligible overhead in priming a node, even temporarily.

The same study [16] also observed that outbound and inbound peer-to-peer traffic through a node at the border of some local network may be imbalanced. This indicates that the location of a node in the underlying network may affect the load it has in the overlay distribution trees which it belongs to. It is an interesting topic for future work to try to model this relation between distribution trees and the underlying overlay network, and its effect on the ideas that we present here.

Plaxton, Rajaraman, and Richa proposed a related scheme that also takes into account the underlying topology when overlaying the distribution network [14]. Other related work focused on balancing load through replication but mainly addressed a different aspect—the layout of the distribution trees (placing caches as "nodes") on top of a larger network (e.g., [1, 11]). A conceptually similar problem is that of placing proxy caches in an existing distribution tree to minimize the average number of hops. This problem was explored by Li et al., who provided a quadratic dynamic programming algorithm [12]. The difference between the objective function we use and the one in [12] is due to a basic difference between the applications we consider and those they consider. Placing caches in a network is a slow manual operation that is better optimized via a centralized calculation. In our setting, nodes already have "cache" capabilities, and the goal is to best utilize resources and to do so in a distributed manner.

## 2. Model.

A *distribution tree* is a rooted tree with flow generated at the leaves and flowing toward the root. We suggest algorithms that would *prime* some nodes in the tree. A primed node can serve flow coming into it, while a nonprimed node only forwards flow to its parent. The goal is to prime as few nodes as possible such that all flow is serviced subject to constraints that we define below. As explained in the introduction, flow corresponds to requests, and *priming* corresponds to configuring the node such that it is able to service requests.

Formally, the input to our problem is the tree $T$ and a function $f : \{v \mid v \text{ is a leaf}\} \rightarrow R_+$ mapping each leaf $v$ to a nonnegative real which describes the flow generated at $v$. We also assume another input parameter, denoted by $C$, which is a number that bounds the maximum amount of flow a node can serve. We also assume that each leaf generates at most $C$ units of flow.

An algorithm that solves the problem outputs an *allocation*, which is a subset of the nodes that are primed. The conditions which the allocation has to satisfy vary in the following two service models. These two models differ by whether a primed node is allowed to forward flow or not.

1. *The must-serve model.* In this model a primed node serves all its incoming flow. An allocation constitutes a solution in this model if, when we cut the tree by removing the edge from each primed node to its parent, then the flow generated at the leaves of each subtree is at most $C$ (and therefore can be served at the root of this subtree). In other words, an allocation is a good solution if, when we process nodes from the leaves up, forwarding all the flow into an unprimed node to its parent, then each time we reach a primed node it has at most $C$ units of flow coming into it.
2. *The serve-or-forward model.* In this model a primed node serves at most $C$ units of the flow coming into it and forwards the rest. Here an allocation constitutes a solution if the following process ends such that the amount of flow reaching the root is at most $C$ if it is primed and 0 if it is not primed. We

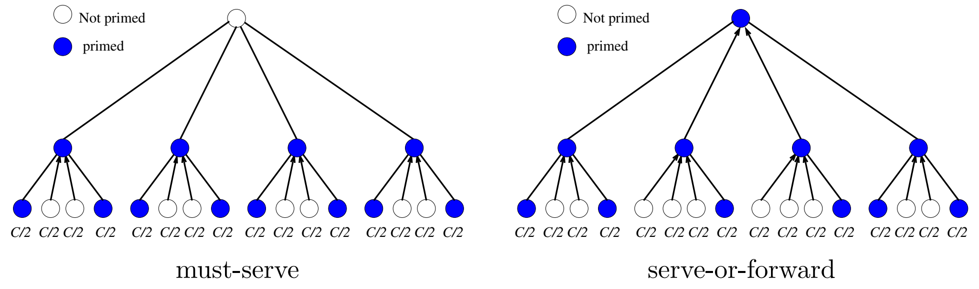


Fig. 2.1. *Example of optimal allocation for the two variants of the problem. The tree has* 16 *leaves, each generating a flow of* $C/2$*; thus the total flow is* $8C$*. Serve-or-forward primes* 11 *nodes and must-serve primes* 12 *nodes.*

process the nodes from the leaves up. When an unprimed node is processed we "forward" all its flow to the parent; when a primed node is processed we assign it to serve all the flow it receives, up to $C$ units, and forward the rest to its parent.

In both models an allocation constitutes a solution if there is an *assignment* of all flow to primed nodes (each unit of flow is assigned to a node on its way to the root) such that the *load* (assigned flow) of each primed node is at most $C$ and the load of other nodes is 0. The *size* of an allocation is the number of nodes it primes. An allocation is *optimal* if it is correct and of minimum size.

The must-serve model is more restrictive than the serve-or-forward model; thus must-serve allocations constitute serve-or-forward allocations. Figure 2.1 illustrates an example flow serviced in the two service models. The serve-or-forward model is appropriate for applications where "servicing" is a more expensive operation than "forwarding" (e.g., when servicing involves intensive computation or large file transfers). The must-serve model is appropriate for applications with low service cost such as simple database lookups. In the must-serve model the traffic traversing each edge (link) is also bounded by $C$, which is important in some applications.

We analyze a static setting where the flow generated by each leaf remains fixed through time.[1] We consider both centralized and distributed algorithms. For distributed algorithms we assume that each node is able to communicate with its parent and children. In the distributed settings, priming decisions of different nodes can depend on each other, and we consider the "convergence" process until the nodes decide on a stable allocation. We measure the convergence time in units of the maximum "reaction time" of a node, that is, the maximum elapsed time until it reacts to a new input.

One of our distributed algorithms, GAP, assumes flow reporting, where each child reports to its parent on the flow generated below it (regardless of whether it is served or not). This reporting process is similar to "hit count" reporting in cache hierarchies.

## 3. What allocation size is necessary?

Let $R$ be the total flow generated in the tree. An obvious lower bound on the required allocation size is $R/C$. This bound can be further refined if we also consider the maximum number of children of a node, which we denote by $\Delta$. The following lemma shows a better bound for the must-serve model.

[1]The algorithms, however, are applicable in a dynamic context where flow changes.

LEMMA 3.1. *For any $C$ and $\Delta > 4$, there are instances such that $\Delta R/(4C)$ primed nodes are necessary in the must-serve model.*

*Proof.* Consider a complete $\Delta$-ary tree of depth $d$, where each leaf generates $2C/\Delta$ flow. Therefore, the total flow is $R = 2C\Delta^{d-1}$.

Consider a two-level subtree of this tree consisting of a node $x$ of depth $d-1$ and its $\Delta$ children which are leaves. At least $\lceil \Delta/2 \rceil$ of the children of $x$ must be primed, since otherwise $x$ receives more than $C$ units of flow. Furthermore, there is an optimal selection that primes exactly $\lceil \Delta/2 \rceil$ of the children of $x$. This is because we can always prime $x$ instead of all but $\lceil \Delta/2 \rceil$ of its primed children without increasing the allocation size.

Let $s(x)$ be the nearest primed ancestor of $x$. (Note that $s(x) = x$ if $x$ is primed.) Since the combined value of the flow of all the unprimed children of $x$ strictly exceeds $C/2$, it follows that if $x \neq y$ are two nodes of the depth $d-1$, then $s(x) \neq s(y)$. (Otherwise $s(x)$ must serve more than $C$ units of flow.) Thus, there is one primed internal node for each node $x$ of depth $d-1$.

Since there are $\Delta^{d-1}$ nodes of depth $d-1$, we obtain that the size of an optimal solution of this instance is $\Delta^{d-1}(1 + \lceil \Delta/2 \rceil) \geq \frac{R\Delta}{4C}$. ☐

A similar bound for the serve-or-forward model is given by the following lemma.

LEMMA 3.2. *For any $C$ and $\Delta > 5$, there are instances such that $\Delta R/(5C)$ storage is necessary with serve-or-forward.*

*Proof.* Consider a complete $\Delta$-ary tree of depth $d$ where all leaf nodes generate $2C/\Delta$ flow and therefore the total flow is $R = 2C\Delta^{d-1}$, as in the proof of Lemma 3.1. The total number of internal nodes is $(\Delta^d - 1)/(\Delta - 1)$, and therefore the total amount of flow which these nodes can serve is $C(\Delta^d - 1)/(\Delta - 1)$. The remaining flow must be served, less efficiently, at primed leaves. The amount of flow remaining for leaves to serve is at least

$$R - C(\Delta^d - 1)/(\Delta - 1) \geq C\Delta^d \left( \frac{2}{\Delta} - \frac{1}{(\Delta - 1)} \right)$$

flow. Each leaf generates $2C/\Delta$ flow, and hence this is the greatest amount of flow any leaf can serve. So we need to prime at least

$$\Delta^d \left( \frac{2}{\Delta} - \frac{1}{(\Delta - 1)} \right) \frac{\Delta}{2} \geq \frac{R\Delta}{5C}$$

leaf nodes for $\Delta \geq 6$. ☐

*Remark.* Note that the proof of Lemma 3.2 applies also to the must-serve model and gives a slightly worse constant than the proof of Lemma 3.1.

## 4. Optimal algorithms.

In this section we present algorithms that find an optimal allocation in both the serve-or-forward and must-serve models. In section 4.1 we present an optimal algorithm for the serve-or-forward model, and in section 4.2 we present an optimal algorithm for the must-serve model.

### 4.1. Serve-or-forward.

The algorithm OPT_SOF works in two phases. In the first phase it processes the nodes from the leaves upward and makes some priming decisions. In the second phase additional nodes are primed throughout the tree. Each node primed in the first phase is assigned $C$ units of flow that are generated below it. Nodes primed in the second phase may be assigned less than $C$ units of flow.[2]

[2]Note that there could be several such mappings for each optimal allocation, including mappings where nodes assigned in the first phase obtain less than $C$ units of flow. We consider particular assignments where this holds.

Our algorithm utilizes the notion of a *residual problem*. In each iteration of the first phase the algorithm primes a node and allocates $C$ units of flow for it to serve. Then it generates a residual instance by transferring children of the primed node $v$ that have remaining unserved flow to $v$'s parent. In the following we denote by $f(v)$ the flow into node $v$ at the "current" residual problem.

ALGORITHM 1 (OPT_SOF).

Phase 1. *While there is a node $v$ other than the root such that $f(v) > C$, do the following. Let $v$ be a deepest nonroot node such that $f(v) > C$, and let $v_1, v_2, \ldots$ be the children of $v$ ordered by increasing flow value, i.e., $f(v_1) \le f(v_2) \le \cdots$.*[3] *Let $j$ be the smallest integer such that $\sum_{i \le j} f(v_i) \ge C$. Prime the node $v$. Assign to $v$ all the flow coming up from $v_1, \ldots, v_{j-1}$ and $C - \sum_{i \le j-1} f(v_i)$ of the flow from leaf descendants of $v_j$ (chosen arbitrarily). Make the children $v_{j+1}, v_{j+2}, \ldots$ of $v$ with their respective flow values $f(v_{j+1}), f(v_{j+2}), \ldots$ children of the parent of $v$. Make $v_j$, with the $f(v_j) - C + \sum_{i<j} f(v_i)$ units of flow generated in its subtree and not served by $v$, a child of the parent of $v$.*[4]

Phase 2. *Let $v$ be the root, and let $v_1, v_2, \ldots$ be the children of the root ordered by increasing flow value, i.e., $f(v_1) \le f(v_2) \le \cdots$. If $f(v) > C$ let $j$ be the smallest integer such that $\sum_{i \le j} f(v_i) \ge C$.*

*Prime the nodes $v_{j+1}, v_{j+2}, \ldots$, and assign each of them to serve all the flow generated in its subtree.*

*If $\sum_{i \le j} f(v_i) > C$, prime also the node $v_j$, and assign it to serve all the flow generated in its subtree.*

*If $f(v) > 0$, prime $v$, and assign it to serve all remaining unserved flow in the tree.*

We can implement OPT_SOF to run in $O(n \log n)$ time, where $n$ is the number of nodes in the tree. To do that we maintain the children of every node in a heap. When processing a node $v$ we do a series of delete-mins on the heap of $v$ until we have accumulated the lightest children whose weight is above $C$. We reinsert the last deleted child back into the heap with its adjusted flow value if the cumulative flow of the deleted children is greater than $C$. Finally, we meld the heap of $v$ with the heap of the parent of $v$.

A distributed implementation of OPT_SOF follows these two phases. In the first "upward" phase, communication occurs from the leaves to the root with nodes primed along the way. In this phase each node sends to its parent the identity and the current flow value of each transferred child. In the second "downward" phase, the root "decides" to prime a subset of its children. Since some of these children are nodes that have been transferred up during the first phase, we then communicate these priming decisions to the nodes themselves. The amount of communication passed when children are transferred could be (in the worst case) of the order of the number of descendants of the transferring node.

We now establish the correctness and optimality of OPT_SOF.

THEOREM 4.1. OPT_SOF *produces a correct allocation of minimum size.*

*Proof.* To establish correctness we have to verify that each primed node is assigned to serve no more than $C$ units of flow and that all flow is served. By the definition of the algorithm it is clear that each node $v$ which gets primed at the first phase is assigned to serve exactly $C$ units of flow. When the first phase ends all children of

[3]If there is more than one node $v$ which is deepest with $f(v) > C$, we break ties arbitrarily.

[4]In the residual problem we eliminate the flow served by $v$. Other than that, the subtrees hanging off of the children of $v$, which are made children of the parent of $v$, remain the same.

the root receive less than $C$ units of flow. Therefore, having a subset of these children serve the flow they generate, and having the root serve the rest, produces a legal solution as long as the subset is large enough that at most $C$ units of flow are served at the root. The subset primed by the second phase clearly satisfies this requirement.

We now show that the allocation produced is of minimum size. Consider an iteration of the first phase. Let $T$ be the network when the iteration starts, let $v$ be the node which is primed during the iteration, and let $A$ be the flow assigned to $v$. Let $T'$ be the residual problem generated from $T$ at the end of the iteration. It is not hard to see that if $T$ has an optimal allocation in which $v$ is primed and serves $A$, then we can obtain an optimal solution to $T$ from any optimal solution of $T'$, by using the allocation of the optimal solution of $T'$ together with priming $v$ and assigning $v$ to serve $A$. Thus by induction on the iterations of the first phase, it suffices to show that the choice of $v$ and $A$ can be extended to an optimal solution of $T$. As a base case of the induction we show that the solution of the residual problem obtained by the second phase is optimal. The optimality of the allocation of the second phase follows since we prime the children of the root with largest incoming flow. Therefore, we prime as small a subset of them as possible so that the root can serve the remaining flow.

It remains to show that $T$ has an optimal solution in which $v$ is primed and serves $A$. Consider any optimal solution $O$ to $T$. We show that we can change it to another solution with the same number of primed nodes in which $v$ serves $A$.

If $v$ is not primed in $O$, then we first change $O$ so $v$ is primed as follows. If $v$ has a primed descendant, then let $v'$ be such a primed descendant where the nodes on the path from $v$ to $v'$ are not primed. Clearly if we prime $v$ instead of $v'$ and have $v$ serve the flow previously served by $v'$, we still have an optimal solution. In case $v$ does not have a primed descendant, let $u$ be the closest ancestor of $v$ which is primed. Since $f(v) > C$ if $u$ serves flow which is not generated below $v$, some ancestor of $u$ serves flow generated below $v$. It follows that by interchanging flow assigned to $u$ with flow assigned to its ancestors, we can change the flow assignment so that all the flow that $u$ serves is generated below $v$. Finally, since all the flow which $u$ serves is generated below $v$, we can prime $v$ instead of $u$ and have it serve the flow that $u$ serves.

Last, we have to show that we can change the allocation and the flow assignment so that $v$ serves $A$. First notice that since the flow reached to each child of $v$ is at most $C$ we may assume that $O$ does not contain prime descendants of $v$ which are not immediate children of $v$. (We can prime a child of $v$ instead of each such primed descendant of $v$ without increasing the allocation.) Furthermore, if $v$ has a primed child $v_1$, and a child $v_2$ which is not primed, such that $f(v_2) > f(v_1)$, we can change $O$ and prime $v_2$ instead of $v_1$. Ancestors of $v$ that served the flow coming up from $v_2$ are now assigned to serve the flow coming up from $v_1$ instead. Since less flow is coming up from $v_1$ this allocation is still legal and hence optimal. So we may assume that the primed children of $v$ are those from which the largest amount of flow originates.

Finally, if $v$ serves $C$ units of flow but this flow is not identical to $A$, we can interchange the flow assigned to $v$ with the flow assigned to its ancestors so that $v$ serves exactly the flow in $A$. If $v$ serves less than $C$ units of flow, we can increase the flow assigned to it so that it serves exactly the flow in $A$ and assign less flow to $v$'s primed child that serves the smallest amount of flow among the children of $v$. ∎

**4.2. Must-serve.** Must-serve instances have a simpler definition of the residual problem: Given an instance and a subset of primed nodes, the residual problem is obtained by eliminating all subtrees rooted at primed nodes along with the flow generated in them.

Unlike the case with serve-or-forward instances, we do not need to consider flow assignments for obtaining the residual instance. The identity of the primed nodes suffices.

We now state the optimal must-serve algorithm, OPT_MS. The algorithm iteratively primes nodes; it considers the residual network after each priming step. Computationally, the algorithm performs one bottom-up pass, but information can propagate one level down from parents back to children during that pass.

ALGORITHM 2 (OPT_MS). *Let $v$ be a deepest node with $f(v) > C$.*

*Order the children of $v$ by decreasing flow $v_1, v_2, \ldots$ with $f(v_1) \geq f(v_2) \ldots$.*

*Let $j$ be the smallest integer such that $\sum_{i \leq j} f(v_i) \geq f(v) - C$. Prime the nodes $v_1, \ldots, v_j$.*

*If there is no $v$ with $f(v) > C$ and positive flow enters the root, then prime the root.*

The correctness and optimality analysis is similar to (and somewhat simpler than) the one performed in the serve-or-forward model.

LEMMA 4.2. *Algorithm* OPT_MS *is correct and finds a minimum-size must-serve allocation.*

*Proof.* Consider a deepest node $v$ with $f(v) > C$. We claim that any solution with the heaviest child of $v$ being unprimed can be transformed to a solution with the same number of primed nodes where the heaviest child of $v$ is primed and with $v$ serving or transferring at most the same amount of flow. The claim concludes the correctness proof since it implies that the repeated step of choosing a deepest node with $f(v) > C$, priming its heaviest child, and focusing on the residual problem is consistent with an optimal solution.

We now prove the claim. Consider an optimal solution where the heaviest child of $v$ is not primed. First observe that no node can receive flow value that exceeds $C$ in a solution; the first primed ancestor that this flow arrives at will have to serve it all, which is a contradiction. Therefore, node $v$ must have at least one primed proper descendant in its subtree. First we show that there is an optimal solution where one of the children of $v$ is primed. Suppose no one of the children of $v$ is primed. It follows that $v$ must have a primed descendant $u$ which is not a child of $v$. Then the child of $v$ which is an ancestor of $u$ can be primed instead of $u$. This child can serve all the flow that $u$ was serving plus additional flow.

Suppose now that the heaviest child of $v$ is not primed and another child $u$ is primed. If we prime the heaviest child and unprime $u$, then the total unserved flow reaching $v$ can only decrease. ∎

We next consider algorithms where nodes make local decisions on when to become primed, by simply observing incoming flow.

## 5. The THRESHOLD algorithm.

The THRESHOLD$(C, \Delta)$ algorithm runs independently at each node. The only communication it receives is the amount of incoming flow. We also assume each node knows $\Delta$, the indegree of its parent node (or some bound on it). THRESHOLD produces an allocation which is good for the must-serve model.

A copy of THRESHOLD running at a node $v$ performs as follows.

ALGORITHM 3 (THRESHOLD$(C, \Delta)$). *Let $f(v)$ be the flow into the node $v$ with respect to the current allocation assuming each primed descendant of $v$ serves all the flow it gets.*

*The node $v$ is primed if and only if*

- *$f(v) > C/\Delta$, or*
- *$v$ is the root and $f(v) > 0$.*

In our model nodes may make their decisions asynchronously. Let $u$ be an ancestor of $v$ such that there is no primed node which is a proper descendant of $u$ and a proper ancestor of $v$ in the current allocation. We assume that if node $v$ decides to change its priming status from primed to unprimed or vice versa, the flow into $u$ changes instantaneously to reflect this change. We also assume that each node runs the algorithm above at least every $t$ seconds. Therefore, if the incoming flow into a node changes so that it crosses the threshold $C/\Delta$, and it remains stable at its new value for at least $t$ seconds, then the priming status of $v$ would change.

In reality there may be a delay from the moment a node $v$ changes its priming status until an ancestor $u$ of $v$ realizes that its incoming flow has changed. Furthermore, this delay may be proportional to the distance between $u$ and $v$. We discuss at the end of this section how this may affect the performance of THRESHOLD.

Initially, the algorithm starts with some configuration of primed nodes (possibly no primed nodes). Since each node makes local decisions that depend on decisions made by the nodes below it, it may change its decision multiple times. We argue that eventually, a stable allocation is reached which is good for the must-serve model. In fact, THRESHOLD$(C, \Delta)$ converges to the same allocation generated by the following iterative centralized algorithm (assuming it starts with an empty allocation).

- Prime all nodes such that the flow generated below them exceeds $C/\Delta$, and the flow generated below each of their children does not exceed $C/\Delta$.
- Truncate the tree by removing these newly primed nodes along with all nodes and flow generated below them, and repeat this process.

Let $N$ be the set containing each node $v$ such that the total flow generated in the subtree of $v$ exceeds $C/\Delta$ and the total flow generated below each child of $v$ is no larger than $C/\Delta$. The centralized algorithm above primes the nodes in $N$ in its first iteration. We claim that THRESHOLD also primes $N$ and unprimes all descendants of nodes in $N$, after at most $2t$ seconds, starting with any allocation. Indeed, regardless of the initial allocation, descendants of nodes in $N$ must see incoming flow of at most $C/\Delta$. Thus, THRESHOLD cannot prime them, and if they started out being primed, then after at most $t$ seconds THRESHOLD will unprime them. Once all descendants of nodes in $N$ are unprimed, each node in $N$ receives flow exceeding $C/\Delta$ so it will become primed in the following $t$ seconds. Once the nodes in $N$ are primed and all their descendants are unprimed, this state will prevail as long as the input flow does not change. By induction one can complete this argument to show that above the nodes of $N$, THRESHOLD also primes exactly the same nodes as the iterative algorithm above when applied to the truncated tree.

Observe that since each child of a node in $N$ sends to its parent at most $C/\Delta$ flow, the total flow that each node in $N$ receives is at most $C$. So when THRESHOLD primes a node in $N$, this node can serve all the flow generated below it, and it transfers 0 flow to its parent. The following lemma follows from this equivalence between these two algorithms.

LEMMA 5.1. THRESHOLD$(C, \Delta)$ *eventually reaches a correct stable allocation. The allocation produced is legal in the must-serve model and thus also applies to the serve-or-forward model. The final allocation is of size at most $R\Delta/C$ (since each primed node serves at least $C/\Delta$ flow).*

We define the *convergence time* of THRESHOLD as the time it takes for it to reach the stable state. To bound the convergence time of THRESHOLD recall that we defined

$t$ to be the maximum time it takes for a node to respond to a change in its incoming flow, where by *responding* we mean a change of state from prime to unprime or vice versa. It is easy to prove by induction on $i$ that after $i * t$ seconds all nodes of depth at least $d-(i-1)$ have reached their stable state. Indeed, after $t$ seconds the nodes of depth $d$ reach their final state, since their incoming flow does not depend on decisions made by other nodes and therefore cannot change. Once the nodes of depth $d$ are stable, the incoming flow into nodes of depth $d-1$ will not change anymore. So after an additional $t$ seconds nodes of depth $d-1$ reach their stable allocation as well. In general, after $k*t$ seconds nodes of depth at least $d-(k-1)$ have reached their stable state, so the flow into nodes of depth $d-k$ will not change anymore, and consequently these nodes of depth $d-k$ reach their stable state during the next $t$ seconds.

How far can THRESHOLD be from the optimal solution? We provide examples where the worst-case ratio approaches $\Delta$ for both the serve-or-forward and must-serve models. We start with the serve-or-forward model. For an integer $i$, construct a binary tree of depth $i$, and then attach $\Delta$ children to each of the $2^i$ leaves of the binary tree. Now suppose that each of the $\Delta 2^i$ leaves of the modified tree generates flow of value $C/\Delta+\epsilon$ ($\epsilon < C/(\Delta 2^i)$). The THRESHOLD algorithm will prime all leaves, and this results in an allocation of size $\Delta 2^i$. The algorithm OPT_SoF will prime all depth-$i$ nodes and the root node of the tree and thus will obtain an allocation of size $2^i+1$. The ratio of the allocation size obtained by THRESHOLD to the optimal allocation of OPT_SoF approaches $\Delta$ as $i$ increases.

For the must-serve model consider a tree of depth one consisting of a node and $\Delta$ children. Among these $\Delta$ children $\Delta-1$ generate $C/\Delta+\epsilon$ flow, and the remaining child generates $C/\Delta-(\Delta-1)\epsilon$ flow. The THRESHOLD algorithm primes all $\Delta-1$ children and the root node, thus producing an allocation of size $\Delta$. The optimal algorithm OPT_MS primes only the root. A family of examples with arbitrarily large values of total flow can be obtained by attaching $\Delta$ children generating flow as defined above to each of the leaves of a binary tree. For small enough $\epsilon$, the THRESHOLD algorithm will prime $(\Delta-1)2^i$ of the leaves and all $2^{i-1}$ nodes of depth $i-1$. The algorithm OPT_MS will prime all depth-$i$ nodes. The ratio between the size of the allocation of THRESHOLD and the size of the allocation of OPT_MS is $\Delta-0.5$.

Observe that any algorithm that takes independent local decisions at each node would not be able to improve on this approximation ratio. Intuitively, since no information is available to a node on what its siblings send to its parent, the algorithm cannot allow sending more than $C/\Delta$ units of flow to the parent and at the same time guarantee that every node receives at most $C$ units of flow.

**5.1. Convergence of THRESHOLD.** We learned that the worst-case final allocation obtained by THRESHOLD is within $\Delta$ of optimal. We now consider the "intermediate" allocations obtained during the convergence process.

Consider a tree that consists of a path of $n$ nodes $v_1,\dots,v_n$ (ordered from the bottom to the root); each path node has $\Delta-1$ leaves attached to it. The $\Delta-1$ leaves at each path node generate together $\epsilon C/\Delta$ flow. (To simplify the presentation we assume that $1/\epsilon$ is integral.) The total flow entering the tree is $R=n\epsilon C/\Delta$. Observe that THRESHOLD will never prime a leaf node; thus only the nodes on the path get primed.

Consider the following possible convergence scenario. (1) Initially, all nodes are not primed; thus the $i$th path node $v_i$ sees $i\epsilon C/\Delta$ flow. (2) Every $v_i$ for $i \geq 1/\epsilon$ receives more than $C/\Delta$ flow and primes itself. (3) Next, all primed nodes above $v_{1+1/\epsilon}$ see only $\epsilon C/\Delta$ flow and unprime themselves. (4) The deepest primed path node $v_{1/\epsilon}$ remains primed from now on, since all nodes below it remain unprimed. (5) The process continues on the new tree obtained after truncating the current tree at $v_{1/\epsilon}$. The new tree has the same structure with $n' = n - 1/\epsilon$ interior path nodes.

It is not hard to see that convergence takes $\epsilon n = R\Delta/C$ such "iterations." The average number of nodes that change state (from primed to unprimed or vice versa) in an iteration is $\Omega(n)$. The largest intermediate allocation occurs at the first iteration and is of size $n - 1/\epsilon = (R\Delta/C - 1)/\epsilon$. That is, there is a factor of $1/\epsilon$ between intermediate and final allocation sizes.

Interestingly, these undesirable properties are not merely an artifact of the "sharp" threshold exhibited by THRESHOLD: a similar convergence pattern with very large size of intermediate allocation and slow convergence time is possible even under a natural modification of THRESHOLD. In this modification we prime a node when the incoming flow exceeds $C/\Delta$ and unprime a node when its flow drops to $L/\Delta$ or below (for some $\epsilon C < L < C$). This variant of THRESHOLD would result in worse-quality final allocations (which can be $C/L$ times the allocation size obtained by the original version of THRESHOLD). The worst-case convergence patterns, however, are similar.

In our model we assumed that the flow incoming into any node $v$ changes instantaneously to reflect any change in the allocation of the descendants of $v$. Clearly the scenario above where intermediate allocations are large used this property of the model. Such bad convergence patterns are less likely to happen in cases where, by the time a node $v$ figures out the value of its incoming flow, the allocation in its subtree is stable or close to stable. We can force this to happen if each node $v$ starts out primed and unprimes itself only when the allocation in its subtree is final and the local decision of THRESHOLD at $v$ is to be unprimed. Notice, however, that the bad convergence pattern of THRESHOLD may reoccur at any time where the input changes even if we manage to avoid it at initialization time. Furthermore, for some applications priming a large number of nodes even temporarily may be undesirable.

Another undesirable property of THRESHOLD occurs when the flow incoming into the leaves changes and THRESHOLD adjusts its allocation to the new flow. It is not hard to see that relatively small changes in the input may trigger large changes in the allocation. In particular, a node $v$ may change its allocation even if the total flow generated in its subtree has not changed. Consider, for example, a node $v$ with three children $v_1$, $v_2$, and $v_3$ that are leaves. Assume $v_1$ generates $2\epsilon$ flow, $v_2$ generates $\frac{C}{\Delta} - \epsilon$ flow, and $v_3$ generates $\frac{C}{\Delta} + \epsilon$ flow. THRESHOLD will prime $v_3$ and $v$. Assume now that the flow changes slightly so that $v_1$ generates 0 flow, $v_2$ generates $\frac{C}{\Delta} + \epsilon$, and $v_3$ still generates $\frac{C}{\Delta} + \epsilon$ flow. Then, THRESHOLD primes $v_2$ and unprimes $v$, even though the total flow generated below $v$ has not changed.

To summarize, we showed that the simple local THRESHOLD algorithm can have certain convergence and instability patterns that may be considered as serious drawbacks for some applications. In section 6 we propose and analyze a different local algorithm, called GAP, that overcomes some of problems of THRESHOLD.

**5.2. Parent priming.** By the definition of THRESHOLD, no node can send to its parent more than $C/\Delta$ units of flow in the final allocation. This requirement is necessary to guarantee that no node serves more than $C$ units of flow in situations where a node has no information on what its siblings send to the parent. As a result, THRESHOLD may converge to an allocation of size $\frac{R\Delta}{C}$. Indeed we showed in section 3 that for some trees and flows an allocation of size about $\frac{R\Delta}{C}$ may be necessary, but for other inputs THRESHOLD may produce an allocation that is off from the optimal by a factor of $\Delta$. In this section we suggest a technique that improves this worst-case approximation ratio of $\Delta$ by a constant factor and may perform even better in practice.

Consider running THRESHOLD using the bound $C' = \Delta C$ rather than $C$. This, of course, can result in a primed node serving up to $\Delta C$ units of flow. To solve this problem, and turn the solution into a valid one, we allow a correcting *parent priming* phase, where nodes that were primed in the first phase are allowed to prime some of their children. Specifically, a node that ends up with more than $C$ units of flow (that is, flow in $(C, \Delta C]$) primes its children, in decreasing order of flow value, until its own flow drops to $C$ or fewer units.

It is easy to see that this additional local phase will always result in a correct allocation since before the parent priming phase each unprimed child was delivering at most $C$ units of flow. The next lemma upper bounds the approximation ratio of an allocation produced using the parent priming technique with respect to the optimal allocation.

LEMMA 5.2. *The approximation ratio of* THRESHOLD *with parent priming is* $(\Delta + 2)^2/(4\Delta)$.

*Proof.* Every node that is primed in the first phase serves at least $C$ (and at most $\Delta C$) units of flow. Consider a node primed in the first phase that serves $\alpha C \geq C$ units of flow. An optimal algorithm must use at least $\alpha$ primed nodes to serve the flow. Our algorithm primes at most $\lceil \Delta(\alpha - 1)/\alpha \rceil$ children and thus uses at most $2 + \Delta(\alpha - 1)/\alpha$ primed nodes to serve the flow. The ratio is thus $(2 + \Delta)/\alpha - \Delta/\alpha^2$. The maximum on the interval $\alpha \in [1, \Delta]$ is obtained at $\alpha = (2\Delta)/(\Delta + 2)$ and is $(\Delta + 2)^2/(4\Delta)$. ◻

## 6. The GAP algorithm.

The GAP algorithm runs locally at each node. Like THRESHOLD, GAP makes its decisions locally. However, in contrast to THRESHOLD, decisions made by each node are independent of the state of other nodes. The only information GAP relies on at each node is the total amount of flow generated below each of its children (regardless of where it is served). Note that if nodes are running THRESHOLD and all start unprimed, then the initial value of the incoming flow to each node is the total flow generated in its subtree. In practice, nodes can propagate this information bottom-up on the distribution tree by periodic reports from children to parents.

Let $F(v)$ be the total flow generated under a node $v$, and let $F_s(v)$ be the flow generated under its heaviest child, that is, $F_s(v) = \max\{F(u) \mid u \in \text{children}(v)\}$. If $v$ is a leaf, then we define $F_s(v) = 0$.[5]

We consider GAP in the must-serve model. GAP is a randomized algorithm which depends on two parameters. The first is $C$, which is also an upper bound on the maximum flow generated by a leaf. In previous sections we used $C$ as an upper bound on the flow a primed node is allowed to serve. Here, since GAP is randomized, we will allow some nodes to serve more than $C$ units of flow. The second parameter of GAP is a fraction $0 \leq \mu \leq 1$. This fraction controls the average number of primed nodes, which increases as $\mu$ decreases. As $\mu$ decreases, the probability that a node will serve more than $C$ units of flow also decreases.

Specifically, $\mu$ determines the expected amount of flow served by a primed node. However, whatever this expectation is, the probability that we deviate from it decreases exponentially. For example, if $\mu = \frac{1}{\Delta}$, then the expected amount of flow that

[5]We use here $F(v)$ to distinguish it from $f(v)$, which we used in the previous section to denote the flow entering node $v$.

a node will serve is no greater than $C$, and the probability that a node will have to serve more than $L * C$ units of flow decreases exponentially with $L$.

ALGORITHM 4 (GAP$(C, \mu)$ algorithm). *At each node $v$, GAP$(C, \mu)$ performs the following.*

1. *If $F(v) < \mu C$, then $v$ is not primed.*
2. *If $F(v) \geq \mu C$, then*
   - *if $F_s(v) < \mu C$, $v$ is primed;*
   - *otherwise, $v$ is primed with probability* $\min\{1, (F(v) - F_s(v))/(\mu C)\}$.
3. *If $v$ is the root, it did not get primed in 2, and it has flow entering it, then prime $v$.*

In our subsequent analysis, we assume that only leaf nodes have flow below $\mu C$. We do that without loss of generality since the operation of GAP on an instance is equivalent to GAP operating on the same instance when maximal subtrees rooted at a node $v$ with $F(v) < \mu C$ are contracted to a single leaf with flow $F(v)$.

We first bound the expected size of the allocation produced by GAP.

LEMMA 6.1. *The expected number of nodes primed by* GAP *at step 2 is at most $2R/(\mu C) - 1$, for $R \geq \mu C$. (The actual expected number of primed nodes is at most one larger because of priming the root, and it is one if $R < \mu C$.)*

*Proof.* Let $M(x)$ be the maximum, over flow trees with flow $x$, of the expected number of primed nodes allocated by GAP in step 2. By the definition of GAP, $M(x)$ satisfies the following. If $x < \mu C$, then $M(x) = 0$. For $x \geq \mu C$ we have that

$$M(x) = \max\Bigg( \{\min\{1, (x - S)/(\mu C)\} + M(S) \mid S < x\}$$

$$\cup \left\{ 1 + M(S_1) + \cdots + M(S_i) \mid \sum_{j=1}^{i} S_j \leq x \text{ and } \forall j\ S_j \geq \mu C \text{ and } i \geq 2 \right\} \Bigg). \tag{6.1}$$

The first set over which we take the maximum takes into account all trees where all children of the root except possibly the largest one get fewer than $\mu C$ units of flows. In this case there cannot be primed descendants of any child of the root except the largest one. The second set takes into account trees where more than one child of the root gets at least $\mu C$ units of flow. In this case by the definition of GAP we always mark the root at step 2.

Using induction on the flow value $x$, and relation (6.1), it is easy to establish that $M(x) \leq 2x/(\mu C) - 1$. ∎

Since GAP is a randomized algorithm, there is a positive probability that nodes are assigned flow that exceeds $C$. The following lemma gives probabilistic bounds on the maximum flow entering a node when we apply GAP. Notice that since each leaf generates at most $C$ units of flow, it is clear that a primed leaf never serves more than $C$ units of flow. So the lemma considers only the maximum flow entering a prime internal node.

LEMMA 6.2. *The expected amount of flow serviced by a primed internal node is at most $\Delta \mu C$. Furthermore, the probability that the serviced flow exceeds $L \Delta \mu C$ is at most $(1 - 1/e)^L$.*

*Proof.* To simplify the calculations, we normalize the flow to be in units of $\mu C$. In these units we first have to show that the expected amount of flow serviced by a primed node is at most $\Delta$.

Consider an internal node $v_0$. Since $v_0$ is internal, $F(v_0) \geq 1$. Consider a child $v_1$ of $v_0$. We claim that the expected value of flow that $v_0$ receives from $v_1$ is at most 1.

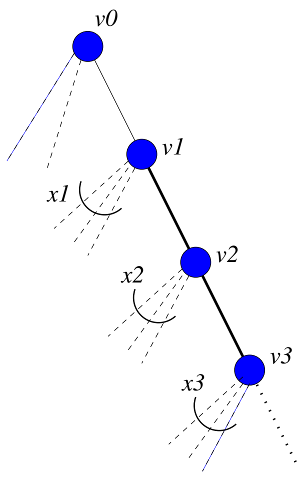


FIG. 6.1. *Illustration for the proof of Lemma* 6.2. *Thick lines represent edges to heaviest child.*

Thus the total expected flow passed to $v_0$ from all its children together is at most $\Delta$.

We now prove the claim. We can assume without loss of generality that $F(v_1) > 1$ (since otherwise the claim trivially follows). We may also assume that $v_1$ is not a leaf, since, if it was, then it is primed and therefore does not contribute any flow to $v_0$. Let $v_2$ be the child of $v_1$ with the largest amount of flow generated below it and in general let $v_{i+1}$ be the child of $v_i$ with the largest amount of flow generated below it, if $v_i$ is not a leaf. We call $v_1, v_2, \ldots, v_\ell$ the *heavy path* hanging from $v_1$. (See Figure 6.1.) Note that $v_\ell$ is a leaf. Let $x_i = F(v_i) - F_s(v_i) = F(v_i) - F(v_{i+1})$ $(i \geq 1)$ be the flow getting into $v_i$ from all its children but $v_{i+1}$. We define a node on the *heavy path* hanging from $v_1$ to be *always-primed* if one of the following conditions holds.

1. $x_i \geq 1$. That is, the flow that enters $v_i$ from children other than $v_{i+1}$ is at least 1.
2. $F(v_i) \geq 1$ and $i = \ell$. That is, $v_i$ is the leaf at the end of the path and it generates at least one unit of flow.
3. $F(v_i) \geq 1$, $i < \ell$, and $F(v_{i+1}) < 1$. Note that this case can happen only if $i = \ell - 1$ since we assume that only leaves can have $F(v) < 1$.

Note that by the definition of GAP an *always-primed* node is primed with probability one. Furthermore, either $v_\ell$ or $v_{\ell-1}$ must be always-primed, so in particular we have at least one always-primed node on the path. If $v_1$ is always-primed, then the claim obviously holds; otherwise, let $k \geq 1$ be the maximum such that $v_k$ is not always-primed. Observe that we must have that $x_i < 1$ for $1 \leq i \leq k$.

Consider a run of GAP, and let $p \leq k$ be the lowest index such that $v_p$ is primed; define $p = k+1$ if there is no primed node before $v_{k+1}$ on the path. By definition, the maximum amount of flow that $v_0$ gets from $v_1$ is $\sum_{j \leq k+1} x_j$. To estimate the expected amount of flow that $v_0$ gets from $v_1$, we define $G_i$, $1 \leq i \leq k$, to be a random variable describing the flow value passed to $v_{i-1}$ from $v_i$. We also denote the expectation of $G_i$ by $g_i$. Since GAP leaves node $v_i$ unmarked with probability $1 - x_i$, we have that $g_k = (1 - x_k)x_k$, and for $i < k$

$$g_i = (1 - x_i)(x_i + g_{i+1}).$$

So, in particular,

$$g_1 = (1-x_1)(x_1 + (1-x_2)(x_2 + (1-x_3)(\cdots + (1-x_{k-1})(x_{k-1} + (1-x_k)x_k)\cdots))).$$

We show by a backward induction on $i$ that $g_i \leq 1$. It is easy to see that $g_k = (1-x_k)x_k \leq 1$. Assume inductively that $g_i \leq 1$; then $g_{i-1} = (1-x_i)(x_i + g_i) \leq (1-x_i)(x_i+1) \leq 1$. For $i = 1$ we obtain that $g_1 \leq 1$ and the claim follows.

We now prove the second part of the lemma. We start with the following claim.

Claim 6.1. *The probability that the value of $G_i$ exceeds $L$ is at most $(1-1/e)^L$ for all $L \geq 0$.*

We prove the claim by downward induction on $i$. The base of the induction is the random variable $G_k$. The outcome is $x_k$ with probability $1-x_k$ and 0 otherwise. The likelihood that the outcome exceeds $L > x_k$ is 0. The likelihood that it exceeds $L \leq x_k$ is $(1-x_k) \leq (1-1/e)^{x_k} \leq (1-1/e)^L$. (We use the tautology $(1-a) \leq (1-1/e)^a$ for $0 \leq a$.)

For the random variable $G_i$ and $L \leq x_i$ we have

$$P[G_i \geq L] = (1-x_i) \leq (1-1/e)^{x_i} \leq (1-1/e)^L.$$

For $L > x_i$ we have the relation

$$P[G_i \geq L] = (1-x_i)P[G_{i+1} \geq L - x_i].$$

From the induction hypothesis we have $P[G_{i+1} \geq L-x_i] \leq (1-1/e)^{L-x_i}$. Substituting the above, we obtain that

$$P[G_i \geq L] \leq (1-x_i)(1-1/e)^{L-x_i} \leq (1-1/e)^L.$$

(The last inequality follows again from the inequality $(1-a) \leq (1-1/e)^a$ for $(0 < a < 1)$.) This completes the proof of the claim.

Claim 6.1 shows that the probability that one child passes more than $L$ units of flow to its parent is at most $(1-1/e)^L$. To complete the proof of the lemma we need to show that when a node has $\Delta$ children then the probability that it receives $\Delta L$ units of flow from all of them is still at most $(1-1/e)^L$. This intuitively should hold since priming decisions at nodes which are unrelated (neither is a descendant of another) are independent. The following claim establishes this fact and finishes the proof of the lemma.

Claim 6.2. *Let $Y_1, \ldots, Y_\Delta$ be independent random variables such that $P(Y_i \geq L) \leq (1-1/e)^L$ for all $L \geq 0$. Let $Z = Y_1 + Y_2 + \cdots + Y_\Delta$. Then $P(Z \geq \Delta L) \leq (1-1/e)^L$ for all $L \geq 0$.*

We prove Claim 6.2 using the notion of stochastic domination between random variables. We say that a random variable $Z$ *stochastically dominates* a random variable $X$ if $P(X \geq a) \leq P(Z \geq a)$ for every $a$. It is known (see, e.g., [17, p. 52]) that if $Z_1, \ldots, Z_k$ are independent, $X_1, \ldots, X_k$ are independent, and $Z_i$ stochastically dominates $X_i$ for every $1 \leq i \leq k$, then $\sum_{i=1}^k Z_i$ stochastically dominates $\sum_{i=1}^k X_i$. For completeness we prove this here.

Since $Z_k$ stochastically dominates $X_k$, we obtain that

$$P\left(\sum_{i=1}^k X_i \geq a\right)$$
$$= \int_{x_1=0}^{\infty}\int_{x_2=0}^{\infty}\cdots\int_{x_{k-1}=0}^{\infty} P(X_1 = x_1)\cdots P(X_{k-1} = x_{k-1})P(X_k \geq a - x_1 - \cdots x_{k-1})$$

$$\leq \int_{x_1=0}^{\infty}\int_{x_2=0}^{\infty}\cdots\int_{x_{k-1}=0}^{\infty} P(X_1 = x_1)\cdots P(X_{k-1} = x_{k-1})P(Z_k \geq a - x_1 - \cdots x_{k-1})$$
$$= P\left(\sum_{i=1}^{k-1} X_i + Z_k \geq a\right).$$

We can now apply the same argument to the variable $X_1, \ldots, X_{k-1}$ and $Z_k$ and show that

$$P\left(\sum_{i=1}^{k-1} X_i + Z_k \geq a\right) \leq P\left(\sum_{i=1}^{k-2} X_i + Z_{k-1} + Z_k \geq a\right).$$

For that we expand $P(\sum_{i=1}^{k-1} X_i + Z_k \geq a)$ as a multiple integral whose variables are the values of $X_1, \ldots, X_{k-2}$ and $Z_k$, and we use the fact that the probability of a tail of $X_{k-1}$ is dominated by the probability of the same tail of $Z_{k-1}$. By repeating this argument $k$ times we finally obtain that

$$P\left(\sum_{i=1}^{k} X_i \geq a\right) \leq P\left(\sum_{i=1}^{k} Z_i \geq a\right).$$

To prove Claim 6.2, let $D$ be an exponential random variable whose cumulative distribution function is $F(x) = P(D \leq x) = 1 - e^{-\lambda x}$ for $x \geq 0$, where $\lambda = -\ln(1 - 1/e)$. Notice that for every $L \geq 0$, $P(Y_i \geq L) \leq P(D \geq L)$, and so $D$ stochastically dominates $Y_i$ for every $1 \leq i \leq \Delta$. (For every $a < 0$, $P(Y_i \geq a) = P(D \geq a) = 0$.) It follows that if $D_1, \ldots, D_\Delta$ are $\Delta$ independent copies of $D$, then $\sum_{i=1}^{\Delta} D_i$ stochastically dominates $\sum_{i=1}^{\Delta} Y_i$. So

$$P\left(\sum_{i=1}^{\Delta} Y_i \geq \Delta L\right) \leq P\left(\sum_{i=1}^{\Delta} D_i \geq \Delta L\right).$$

To finish the proof we observe [6] that the distribution of the sum of $\Delta$ exponential random variables with parameter $\lambda$ is $Gamma(\Delta, \lambda)$ whose density is

$$g(x) = \frac{\lambda e^{(-\lambda x)}(\lambda x)^{(\Delta-1)}}{(\Delta - 1)!}.$$

Next we show that $g(a\Delta) \leq f(a)$ for every $a \geq 0$, where $f(x) = \lambda e^{-\lambda x}$ is the density function of the exponential random variable $D$. Then the lemma follows since $P(\sum_{i=1}^{\Delta} D_i > \Delta L) \leq P(D > L)$.

To see that $g(a\Delta) \leq f(a)$ we substitute $a\Delta$ and $a$ into $g(x)$ and $f(x)$, respectively, and obtain that we have to show that

$$\frac{\lambda e^{(-\lambda a\Delta)}(\lambda a\Delta)^{(\Delta-1)}}{(\Delta - 1)!} \leq \lambda e^{-\lambda a}.$$

Dividing both sides by $\lambda e^{-\lambda a} e^{(-\lambda a\Delta)}$, we obtain the equivalent inequality

$$\frac{e^{(\lambda a)}(\lambda a\Delta)^{(\Delta-1)}}{(\Delta - 1)!} \leq e^{\lambda a\Delta}. \tag{6.2}$$

Using the Taylor expansion of the exponential function, we obtain that (6.2) is equivalent to

$$\left(1+\lambda a+\frac{(\lambda a)^2}{2!}+\cdots\right)\frac{(\lambda a\Delta)^{(\Delta-1)}}{(\Delta-1)!}\leq\left(1+\lambda a\Delta+\frac{(\lambda a\Delta)^2}{2!}+\cdots\right)\ . \tag{6.3}$$

Since

$$\frac{1}{k!}\leq\frac{\Delta^k}{\Delta(\Delta+1)(\Delta+2)\cdots(\Delta+(k-1))},$$

we obtain that (6.3) holds since term $i$ of the series on the left-hand side is not larger than term $i+(\Delta-1)$ of the series on the right-hand side. □

The analysis of GAP implies that if we set $\mu=\frac{1}{c\Delta\log(R\Delta/C)}$ for some small constant $c$, then there is a constant probability that the load served by any primed node is at most $C$. We next show how we can use $\mu=1/(\Delta+1)$ together with iterative application of GAP in order to obtain better trade-off.

**6.1. Applying GAP iteratively.** GAP can be applied iteratively as follows. We apply the first iteration of GAP as described before. After the iteration ends, if we chop off primed nodes from their parents, we get a forest. We apply the next iteration only to trees of the forest where the root serves more than $C$ units of flow. When an iteration ends we further partition the forest by chopping off primed nodes and apply the next iteration only to trees where the root serves more than $C$ units of flow. The algorithm terminates when all primed nodes serve fewer than $C$ units of flow. We show that for an appropriate choice of $\mu$ the algorithm terminates, and we analyze the expected number of iterations until it stops.

We can apply this algorithm distributively by passing messages between nodes as follows. For each unprimed node $v$ we define $F_i(v)$ to be the amount of flow generated in the subtree rooted by $v$ at iteration $i$. For a node $v$ which was primed at iteration $j<i$ we define $F_i(v)=0$. Each unprimed node $v$ locally waits until it knows $F_i(v)$ for increasing values of $i$. When $v$ knows $F_i(v)$ it runs GAP locally and decides whether it gets primed at iteration $i$.

Each node $v$ can calculate $F_i(v)$ if nodes report their $F_i()$ values to their parents. If $v$ was primed at iteration $j<i$, then no flow travels from $v$ to its parent at iteration $i$. So in this case $v$ reports to its parent that $F_i(v)$ is 0. Otherwise, $v$ waits for its children to report their $F_i()$ values. Once $v$ gets all these values, then $F_i(v)$ is $\sum F_i(u)$, where the sum is over all children $u$ of $v$. Node $v$ then reports $F_i(v)$ to its parent.

When we apply this distributed implementation, unprimed nodes continue to apply GAP in subsequent iterations although they may already have a primed ancestor which serves fewer than $C$ units of flow. Therefore, to make our distributed algorithm complete, each node $v$ which is primed at iteration $i$ and $F_{i+1}(v)\leq C$ sends a message downward to its descendants announcing that they may stop running GAP. A node $v$ that receives a message that it has an ancestor that got primed at iteration $i$ and serves no more than $C$ does the following.

1. Node $v$ propagates the message further down to its descendants.
2. If $v$ got primed at iteration $j$ and $i<j$, then $v$ unprimes itself.
3. Node $v$ does not have to run GAP or propagate $F_j(v)$ in any iteration $i<j$.

Our key lemma below shows that if we apply GAP with $\mu=\frac{1}{\Delta+1}$, then on average a constant fraction of the flow is served by primed nodes that serve fewer than $C$ units of flow. This lemma implies the following theorem about the performance of iterative GAP.

THEOREM 6.3. *Iterative* GAP *with* $\mu = \frac{1}{\Delta+1}$ *stops after* $O(\log(R/C))$ *iterations with probability at least* $1 - (\frac{C}{R})^c$, *where c is some constant. The expected size of the allocation that iterative* GAP *produces is* $O(R\Delta/C)$.

*Proof.* By Lemma 6.4, at each iteration of GAP, the probability that a unit flow is served by a primed node which serves fewer than $C$ units of flow is at least $p = 1/e^2$. Each such unit of flow does not participate in subsequent iterations. So the probability that a unit of flow participates at iteration $i+1$ is at most $(1-p)^i$. For $i = \frac{1}{p} a \ln(R/C)$ this probability is at most $(\frac{C}{R})^a$. Let $X$ be a random variable which equals the expected amount of flow participating at iteration $i+1$. Multiplying our bound on the probability that a unit of flow participates at iteration $i+1$ by the total amount of flow, we obtain that $E(X) = R * (\frac{C}{R})^a = C(\frac{C}{R})^{a-1}$. By Markov's inequality the probability that the amount of flow that participates at iteration $i+1$ exceeds $C$ is bounded by

$$\text{Prob}(X \geq C) \leq \frac{E(x)}{C} = \frac{C(\frac{C}{R})^{a-1}}{C} = \left(\frac{C}{R}\right)^{a-1} .$$

Clearly if the total flow which participates at iteration $i+1$ is not larger than $C$, then iteration $i+1$ is the last. So we obtained that, for $i = \frac{1}{p} a \ln(R/C)$, iteration $i+1$ is the last, with probability at least $1 - \left(\frac{C}{R}\right)^{a-1}$. If we define $c = a-1$, the first part of the theorem follows.

By Lemma 6.1, the expected number of primed nodes in each iteration is bounded by $(\Delta+1)/C$ times the expected flow value at that iteration. Since the expected flow value decreases geometrically by a factor of at least $1-p$ from one iteration to the next, we obtain an allocation of expected size $O(R\Delta/C)$. ∎

We now prove our key lemma saying that a unit of flow does not go through to the next iteration of GAP with some constant probability.

LEMMA 6.4. *Consider a unit of flow generated at a leaf. When nodes are primed according to* GAP, *then with probability at least* $1/e^2$ *the primed node serving the flow (the deepest primed node on the path from the leaf to the root) serves a total of at most* $(\Delta+1)\mu C$ *units of flow.*

*Proof.* In the proof of Lemma 6.2 (using Claim 6.2) we proved that the probability that the $\Delta$ children of a node $x$ transfer more than $\Delta L \mu C$ units of flow to $x$ is at most $(1-1/e)^L$. It is easy to see that the same argument in fact shows that any set of $M$ independent nodes (that is, a set of nodes such that neither is an ancestor of the other but they are not necessarily siblings) transfer to their parents $ML\mu C$ units of flow with probability at most $(1-1/e)^L$. We shall repeatedly invoke this observation in the proof.

Consider a unit of flow generated at a leaf and its path to the root. Let $v$ be the deepest node on the path with $F(v) \geq \mu C$. We split the rest of the proof into the following cases.

*Case* 1. For every child $u$ of $v$, $F(u) < \mu C$. In this case $v$ is primed and therefore serves our flow-unit. By the observation above the probability that $v$ receives more than $\Delta \mu C$ flow from its $\Delta$ children is at most $1 - 1/e$.

*Case* 2. Node $v$ has a child $u$ such that $F(u) \geq \mu C$. Let $w$ be the heaviest child of $v$, and let $v_0$ be the closest ancestor of $v$ which is not a heaviest child of its parent, or the root in case every ancestor of $v$ is a heaviest child of its parent. (Note that $v_0 = v$ in case $v$ is not the heaviest child of its parent.) Denote by $v_0, v_1, \ldots, v_i = w$ the nodes on the path from $v_0$ to $w$. Note that $v_{i-1} = v$. Let $x_j$, $0 \leq j < i$, be the flow generated at the children of $v_j$ other than $v_{j+1}$. (See Figure 6.1.)

*Subcase* 2a. $\sum_{j=0}^{i-1} x_j \leq \mu C$. Assume first that $v_0$ is not the root and let $p(v_0)$ be the parent of $v_0$. Since $F(v_0) \geq F(v_i) \geq \mu C$ and $v_0$ is not the heaviest child of $p(v_0)$, we obtain that $F(p(v_0)) - F_s(p(v_0)) \geq \mu C$, so $p(v_0)$ is primed by the definition of GAP. Therefore, our unit flow is served either by $p(v_0)$ or by some $v_j$, where $0 \leq j \leq i-1$.

If our unit flow is served by $v_j$, where $0 \leq j \leq i-1$, then the probability that $v_j$ serves more than $\Delta\mu C$ units of flow is at most the probability that $v_i$ passes up more than $(\Delta-1)\mu C$ units of flow. That happens with probability at most $(1-1/e)^{\Delta-1}$ by Claim 6.1.

If our unit flow is served by $p(v_0)$, then the probability that $p(v_0)$ serves more than $(\Delta+1)\mu C$ units of flow is at most the probability that its $\Delta - 1$ children other than $v_0$ and the node $v_i = w$ transfer to their parents more than $\Delta\mu C$ units of flow. By the observation above this probability is at most $1 - 1/e$.

If $v_0$ is the root, then clearly our unit of flow is served by $v_j$ for some $0 \leq j \leq i-1$. The probability that $v_j$ serves more than $\Delta\mu C$ units of flow is at most the probability that $v_i$ passes up more than $(\Delta-1)\mu C$ units of flow. That happens with probability at most $(1-1/e)^{\Delta-1}$.

*Subcase* 2b. $\sum_{j=0}^{i-1} x_j > \mu C$. Let $k$ be the maximum such that $\sum_{j=k}^{i-1} x_j > \mu C$. If $v_k$ is the root or $x_k > \mu C$, then clearly one of $v_k, \ldots, v_{i-1}$ serves our unit flow. Otherwise, since by Claim 6.1 node $v_{k-1}$, the parent of $v_k$, receives from $v_k$ more than $\mu C$ flow with probability at most $1 - 1/e$, it follows that with probability at least $1/e$ one of $v_k, \ldots, v_{i-1}$ is primed.

Let $v_j$ be the deepest primed node among $v_k, \ldots, v_{i-1}$ given that one of these nodes is primed. The probability that $v_j$ serves more than $(\Delta+1)\mu C$ units of flow is at most the probability that its $\Delta - 1$ children other than $v_{j+1}$ and $v_i$ transfer to their parents more than $\Delta\mu C$ units of flow. By the observation above this probability is at most $1 - 1/e$.

To summarize we obtained that with probability at least $1/e$ one of $v_k, \ldots, v_{i-1}$ is primed, and assuming that one of $v_k, \ldots, v_{i-1}$ is primed, the deepest node among them which is primed served fewer than $(\Delta+1)\mu C$ units of flow with probability at least $1/e$. $\square$

**Acknowledgment.** We would like to thank the referees for many valuable comments that helped us make the presentation much clearer.

## REFERENCES

[1] B. Awerbuch, Y. Bartal, and A. Fiat, *Distributed paging for general networks*, J. Algorithms, 28 (1998), pp. 67–104.

[2] Clip2.com, *The Gnutella Protocol Specification* v0.4, 2000, http://www9.limewire.com/developer/gnutella_protocol_0.4.pdf.

[3] E. Cohen and H. Kaplan, *Aging through cascaded caches: Performance issues in the distribution of Web content*, in Proceedings of the ACM Special Interest Group on Data Communication Conference, ACM, New York, 2001, Computer Communication Review, 41 (4, special issue) (2001), pp. 41–53.

[4] E. Cohen and H. Kaplan, *Balanced-replication algorithms for distribution trees*, in Proceedings of the 10th Annual European Symposium, Lecture Notes in Comput. Sci. 2461, Springer-Verlag, New York, 2002, pp. 297–309.

[5] L. Fan, P. Cao, J. Almeida, and A. Z. Broder, *Summary cache: A scalable wide-area web cache sharing protocol*, IEEE/ACM Trans. Networking, 8 (2000), pp. 281–293.

[6] W. Feller, *An Introduction to Probability Theory and Its Applications, Vol.* 2, John Wiley and Sons, New York, 1971.

[7] S. Gadde, J. Chase, and M. Rabinovich, *A taste of crispy squid*, in Proceedings of the Workshop on Internet Server Performance, Madison, WI, 1998, http://www.cs.wisc.edu/~cao/WISP98-program.html.

[8] Napster Inc., *The Napster Homepage*, http://www.napster.com/, 2001.

[9] D. Karger, E. Lehman, T. Leighton, M. Levine, D. Lewin, and R. Panigraphy, *Consistent hashing and random trees: Distributed caching protocols for relieving hot spots on the World Wide Web*, in Proceedings of the 29th Annual ACM Symposium on Theory of Computing, ACM, New York, 1997, pp. 654–663.

[10] D. Karger, A. Sherman, A. Berkheimer, B. Bogstad, R. Dhanidina, K. Iwamoto, B. Kim, L. Matkins, and Y. Yerushalmi, *Web caching with consistent hashing*, in Proceedings of the 8th International Conference on the World Wide Web, Elsevier, North–Holland, Amsterdam, 1999, pp. 1203–1213.

[11] M. R. Korupolu, C. G. Plaxton, and R. Rajaraman, *Placement algorithms for hierarchical cooperative caching*, J. Algorithms, 38 (2001), pp. 260–302.

[12] B. Li, M. Golin, G. Italiano, X. Deng, and K. Sohraby, *On the optimal placement of web proxies in the internet*, in Proceedings of the IEEE Conference on Computer Communications, New York, 1999, pp. 1282–1290.

[13] R. Malpani, J. Lorch, and D. Berger, *Making World Wide Web caching servers cooperate*, in Proceedings of the Fourth International World Wide Web Conference, Boston, MA, 1995, http://bmrc.berkeley.edu/research/publications/1995/138/paper-59.html.

[14] C. Plaxton, R. Rajaraman, and A. W. Richa, *Accessing nearby copies of replicated objects in a distributed environment*, Theory Comput. Syst., 32 (1999), pp. 241–280.

[15] S. Ratnassamy, P. Francis, M. Handley, R. Karp, and S. Shenker, *A scalable content-addressable network*, in Proceedings of the ACM Special Interest Group on Data Communication Conference, ACM, New York, 2001, Computer Communication Review, 41 (4, special issue) (2001), pp. 161–172.

[16] S. Saroiu, K. P. Gummadi, R. Dunn, S. D. Gribble, and H. M. Levy, *An analysis of internet content delivery systems*, in Proceedings of the Fifth Symposium on Operating Systems Design and Implementation, Boston, MA, 2002, ACM SIGOPS Operating Systems Review, 36 (special issue) (2002), pp. 315–327.

[17] C. Scheideler, *Probabilistic Methods for Coordination Problems*, HNI-Verlagsschriftenreihe 78, University of Paderborn, Paderborn, Germany, 2000.

[18] I. Stoica, R. Morris, D. Karger, H. Frans Kaashoek, and M. ad Balakrishnana, *Chord: A scalable peer-to-peer lookup service for internet applications*, in Proceedings of the ACM Special Interest Group on Data Communication Conference, ACM, New York, 2001, Computer Communication Review, 41 (4, special issue) (2001), pp. 149–160.

SIAM J. COMPUT.
Vol. 34, No. 1, pp. 248–259

# MULTIEMBEDDING OF METRIC SPACES[*]

YAIR BARTAL[†] AND MANOR MENDEL[†]

**Abstract.** Metric embedding has become a common technique in the design of algorithms. Its applicability is often dependent on how large the embedding's distortion is. For example, embedding finite metric space into trees may require linear distortion as a function of the size of the metric. Using probabilistic metric embeddings, the bound on the distortion reduces to logarithmic in the size of the metric.

We make a step in the direction of bypassing the lower bound on the distortion in terms of the size of the metric. We define "multiembeddings" of metric spaces, in which a point is mapped onto a set of points, while keeping the target metric of polynomial size and preserving the distortion of paths. The distortion obtained with such multiembeddings into ultrametrics is at most $O(\log \Delta \log\log \Delta)$, where $\Delta$ is the *aspect ratio* of the metric. In particular, for expander graphs, we are able to obtain *constant* distortion embeddings into trees, in contrast with the $\Omega(\log n)$ lower bound for all previous notions of embeddings.

We demonstrate the algorithmic application of the new embeddings for two optimization problems: *group Steiner tree* and *metrical task systems*.

**Key words.** metric embeddings, group Steiner tree, metrical task systems

**AMS subject classifications.** 68W25, 68R10

**DOI.** 10.1137/S0097539703433122

## 1. Introduction.

Finite metric spaces and their analysis play a significant role in the design of combinatorial algorithms. Many algorithmic techniques were introduced in recent years concerning and using metric spaces and their approximate embedding in other spaces; see the surveys [20, 21] for an overview of this topic.

DEFINITION 1.1. *An embedding of a metric space $M = (V_M, d_M)$ into a metric space $N = (V_N, d_N)$ is a mapping $\phi : V_M \to V_N$. The embedding is called* noncontractive *if for all $u, v \in V_M$, $d_M(u, v) \le d_N(\phi(u), \phi(v))$ and has distortion at most $\alpha$ if, in addition, for all $u, v \in V_M$, $d_N(\phi(u), \phi(v)) \le \alpha \cdot d_M(u, v)$. A noncontractive embedding whose distortion is at most $\alpha$ is called an $\alpha$-embedding.*

The general framework for applying metric embeddings in optimization problems is to embed a given metric space into a metric space from some "nice" family and then apply an algorithm for that space. As a result, the approximation ratio increases by a factor equal to the embedding's distortion.

Among others, embeddings into low dimensional normed spaces [12, 24] as well as probabilistic embeddings into trees [2, 3, 15, 4] have many algorithmic applications. In both cases the distortions of the embeddings are logarithmic in the size of the metric. Unfortunately, there is a matching lower bound on the distortion of these embeddings as well, which sets a limit to their applicability. This paper presents a partial remedy for this problem.

[*]Received by the editors August 9, 2003; accepted for publication (in revised form) August 2, 2004; published electronically October 28, 2004. A preliminary version of this paper appeared in [10].

http://www.siam.org/journals/sicomp/34-1/43312.html

[†]School of Computer Science, Hebrew University, Jerusalem 91904, Israel (yair@cs.huji.ac.il, mendelma@gmail.com). The work of the first author was supported in part by a grant from the Israeli Science Foundation (195/02). The work of the second author was supported by the Landau Center and a grant from the Israeli Science Foundation (195/02).

Tree metrics, and in particular ultrametrics, seem a natural choice as a target class of "simple" metric spaces. Unfortunately, standard embedding is not useful when the target space is a tree metric. Embedding arbitrary metric spaces into trees requires distortion linear in the size of the metric space [27]. Probabilistic embedding [2] provides a way to bypass this problem.

DEFINITION 1.2 (probabilistic embeddings). *A metric space $M = (V_M, d_M)$ is* $\alpha$-probabilistically embedded *in a set of metric spaces $\mathcal{S}$ if there exists a distribution $\mathcal{D}$ over $\mathcal{S}$ and, for every $N \in \mathcal{S}$, a noncontractive embedding $\phi_N : V_M \to V_N$, such that for all $u, v \in V_M$, $\mathbb{E}_{N \in \mathcal{D}}[d_N(\phi_N(u), \phi_N(v))] \leq \alpha \cdot d_M(u, v)$.*

Using probabilistic embeddings, it is possible to obtain much better bounds on the distortion [1, 2, 3, 15, 4]. The following bound is shown in [15, 4].

THEOREM 1.3. *Any metric space on $n$ points can be $O(\log n)$ probabilistically embedded in a set of $n$-point ultrametrics. Moreover, the distribution can be sampled efficiently.*

Theorem 1.3 found many algorithmic applications in approximation algorithms, online algorithms, and distributed algorithms; see, for example, [2, 17, 16, 23, 7]. The bound on the distortion in Theorem 1.3 is tight even for probabilistic embeddings into tree metrics for which there is an $\Omega(\log n)$ lower bound [2].

Theorem 1.3 was originally formulated for a class of metric spaces defined by the following natural generalization of ultrametrics.

DEFINITION 1.4 (after [2]). *For $k \geq 1$, a* $k$-hierarchically well-separated tree *($k$-HST) is a metric space defined on the leaves of a rooted tree $T$. To each vertex $u \in T$ there is associated a label $\Delta(u) \geq 0$ such that $\Delta(u) = 0$ if and only if $u$ is a leaf of $T$. The labels are such that if a vertex $v$ is a child of a vertex $u$, then $\Delta(v) \leq \Delta(u)/k$. The distance between two leaves $x, y \in T$ is defined as $\Delta(\mathrm{lca}(x, y))$, where $\mathrm{lca}(x, y)$ is the least common ancestor of $x$ and $y$ in $T$.*

The definition of a finite ultrametric is the same as a 1-HST. Any $k$-HST is therefore, in particular, an ultrametric, and any finite ultrametric can be $k$-embedded in some $k$-HST [3]. We can therefore restrict our attention to ultrametrics, while all results generalize to $k$-HSTs.

The main contribution of this paper is in offering a new type of metric embedding that makes it possible to bypass lower bounds for the standard and even probabilistic metric embeddings. There are two key observations that lead to this new type of embedding. The first is that in some applications it is natural to match a point onto a set of points in the target metric space. Motivated by two applications of Theorem 1.3—the *group Steiner tree problem* (henceforth, GST) and the *metrical task systems problem* (henceforth, MTS)—we propose the following definition.

DEFINITION 1.5 (multi embedding). *A multiembedding of $M$ in $N$ is a partial surjective function $\mathsf{f}$ from $N$ on $M$; i.e., each point $x \in M$ is embedded into a non-empty set $\mathsf{f}^{-1}(x)$. Points in $\mathsf{f}^{-1}(x)$ are called* representatives *of $x$ in $N$.*

The role of $\mathsf{f}^{-1}$ in Definition 1.5 is analogous to the role of $\phi$ in the Definitions 1.1 and 1.2 of embedding and probabilistic embedding. Another way to define multiembedding is by $\phi : M \to 2^N$, in which $\phi(u) \cap \phi(v) = \emptyset$ for every $u \neq v$. In our notation we have $\phi(x) = \mathsf{f}^{-1}(x)$. Since the $\mathsf{f}$ notation will be more convenient, henceforth we will use it exclusively.

The second observation is that for many applications, including those mentioned above, there is no need to approximate the original distance for every pair of representatives. What is really needed is that every path in the original space be approximated well by *some* path in the target space.

A path in a metric space is an arbitrary finite sequence of points in the space. The length of a simple path $p = \langle u_1, u_2, \ldots, u_m\rangle$ in a metric space $M = (V, d)$ is defined as $\ell(p) = \sum_{i=1}^{m-1} d(u_i, u_{i+1})$.

DEFINITION 1.6 (path distortion). *A multiembedding* $\mathsf{f}$ *of* $M$ *in* $N$ *is called* noncontractive *if for any* $u, v \in N$, $d_N(u, v) \geq d_M(\mathsf{f}(u), \mathsf{f}(v))$. *The path-distortion of a noncontractive multiembedding of* $M$ *in* $N$, $\mathsf{f} : N \to M$, *is the infimum over* $\alpha$, *for which any path* $p = \langle u_1, u_2, \ldots, u_m\rangle$ *in* $M$ *has a path* $p' = \langle u'_1, u'_2, \ldots, u'_m\rangle$ *in* $N$ *such that* $\mathsf{f}(u'_i) = u_i$ *and* $\ell(p') \leq \alpha \cdot \ell(p)$.

*A multiembedding whose path-distortion is at most* $\alpha$ *is called an* $\alpha$*-path embedding.*

A crucial parameter for multiembeddings is the size of the target space $\Gamma$. In general, it will be desirable that $\Gamma$ be polynomial in the size of the source space. In fact, if $\Gamma = \infty$, then there is a simple 1-path embedding of any finite metric space by trees: Take all finite paths, convert each path to a simple path (by duplicating points, if necessary), and put them under a single root with an edge of length half the diameter. This motivates a study of the trade-off between $\Gamma$ and the path distortion of arbitrary metric spaces by tree metrics. In section 4 we study path embedding of expander graphs and the hypercube into tree metrics. We show, e.g., that $n$-point Ramanujan graphs have 3-path embedding into tree metrics of size $\Gamma(n) \leq \text{poly}(n)$. This is in sharp contrast to the status of expander graphs for previous notions of embeddings, for which they are considered "worst case" examples with $\Omega(\log n)$ distortion [24]. These results directly imply nearly tight results on the approximation ratio for GST on expander graphs and hypercubes.

We consider multiembeddings when the class of target metric spaces are ultrametrics. First, we observe that probabilistic embedding into ultrametrics directly implies a bound for path embedding by putting all the trees in $\mathcal{S}$ (the set used in the probabilistic embedding) under a common new root. This results in an $\alpha$-path embedding into an ultrametric of size $|\mathcal{S}|n$. Using the bound of [13] on the number of ultrametrics needed in Theorem 1.3, we obtain an $O(\log n)$ path embedding into an ultrametric of size $O(n^2 \log n)$.[1]

An important parameter of the metric spaces appearing in practice is the aspect ratio of the metric, which is the ratio between the diameter and minimum nonzero distance in the metric space. It will be convenient for us to assume that the minimum distance is 1, and so the aspect ratio becomes the diameter. It turns out that the aspect ratio of the metric plays a significant role in the path distortion of multiembeddings. In section 3 we prove the following result.

THEOREM 1.7. *Fix* $\beta > 1$. *For any metric space* $M = (V, d)$ *on* $|V| = n$ *points and aspect ratio* $\Delta$, *there exists an efficiently constructible multiembedding into an ultrametric of size* $n^\beta$, *whose path distortion is at most*

$$O_\beta(\min\{\log n \cdot \log\log n,\ \log \Delta \cdot \log\log \Delta\}).$$

Our construction beats the probabilistic embedding-based constructions on metrics with small aspect ratio. Expander graphs are examples where a lower bound of $\Omega(\Delta)$ exists on probabilistic embedding using trees [24].

[1]In a preliminary version of this paper [10], we also introduced the notion of *probabilistic multiembedding.* Using that notion we were able to show probabilistic multiembedding into ultrametrics of polynomial size and path distortion $O(\log n \log\log\log n)$. Since an $O(\log n)$ bound now follows from Theorem 1.3 [15, 4], we have decided to drop the probabilistic multiembedding result from this version of the work.

The constructions of multiembeddings are in a sense dual to Ramsey-type theorems for metric spaces [6, 8], where the goal is to find a large subset which is well approximated by some ultrametric.

We also provide a simple example in which Theorem 1.7 is almost tight: Any $\alpha$-path embedding into ultrametrics of a simple unweighted path of length $n$ has $\alpha = \Omega(\log n)$. It follows, in particular, that any $\alpha$-path embedding into ultrametrics of the metric defined by an unweighted graph of diameter $\Delta$ has $\alpha = \Omega(\log \Delta)$. Path embedding is motivated by two intensively studied algorithmic minimization problems: GST and MTS, mentioned above. For both, the best known algorithms use probabilistic embedding into trees/ultrametrics. In section 2 we prove that in order to reduce these problems to other metric spaces it is sufficient to use path embedding. We therefore achieve improved algorithms for these problems whenever the path embedding distortion beats that of probabilistic embedding, and in particular, when the underlying metric is of small aspect ratio.

## 2. Applications.

In this section we define MTS and GST and show that path distortion of multiembeddings reduces these problems to similar problems with different underlying metrics.

MTS [11] was introduced as a framework for many online minimization problems. An MTS on metric space $M = (S, d)$, $|S| = n$, is defined as follows. A "system" has a set of $n$ possible internal states $S$. It receives a sequence of *tasks* $\sigma = \tau_1\tau_2 \cdots \tau_m$. Each task $\tau$ is a vector $\tau : S \to \mathbb{R}^+ \cup \{\infty\}$ of nonnegative costs for serving $\tau$ in each of the internal states of the system. The system may switch states (say, from $u$ to $v$), paying a cost equal to the distance $d(u, v)$ in $M$, and then pays the service cost $\tau(v)$ associated with the new state. The major limiting factor for the system is the requirement to process the sequence in an online fashion, i.e., serving each task without knowing the future tasks.

As is customary in the analysis of online algorithms, MTS is analyzed using the notion of a *competitive ratio*. A randomized online algorithm $A$ is called $r$-competitive if there exists some constant $c$ such that for any task sequence $\sigma$, $\mathbb{E}[\mathrm{cost}_A(\sigma)] \le r \cdot \mathrm{cost}_{\mathrm{OPT}}(\sigma) + c$, where $\mathrm{cost}_A(\sigma)$ is the random variable of the cost for serving $\sigma$ by $A$, and $\mathrm{cost}_{\mathrm{OPT}}(\sigma)$ is the optimal (offline) cost for serving $\sigma$. The current best online algorithm for the MTS problem in $n$-point metric spaces is $O(\log^2 n \log\log n)$-competitive [16, 15] (an improvement of [5, 3]). Both papers [5, 16] actually solve the MTS problem for ultrametrics, and then reduce arbitrary metric spaces to ultrametrics using Theorem 1.3. We next show that path embedding suffices, as follows.

PROPOSITION 2.1. *Assume that a metric space $M$ is $\alpha$-path embedded in $N$. Assume also that $N$ has an $r$-competitive MTS algorithm. Then there is an $\alpha r$-competitive algorithm for $M$.*

*Proof.* We construct an online algorithm $\mathcal{A}$ for $M$ as follows: Let $\mathcal{A}_N$ be an $r$-competitive online algorithm for $N$, and $\mathsf{f} : N \to M$ an $\alpha$ path embedding of $M$ in $N$. The task sequence $\sigma$ is translated to a task sequence $\sigma^N$ for $N$ task by task as follows. A task $\tau$ for $M$ is translated into a task $\tau^N$ for $N$ such that $\tau^N(u') = \tau(\mathsf{f}(u'))$. $\mathcal{A}$ maintains the invariant that if $\mathcal{A}_N$ is in state $v'$, then $\mathcal{A}$ is in state $\mathsf{f}(v')$.

It is easy to verify that $\mathrm{cost}_{\mathcal{A}}(\sigma) \le \mathrm{cost}_{\mathcal{A}_N}(\sigma^N)$, since the service costs are the same, and the distances in $N$ are larger. Consider $\mathrm{OPT}(\sigma)$; it defines a path $p$ serving $\sigma$ in $M$. Thus there exists a path $p^N$ as in the statement of Definition 1.6. The path $p^N$ is the way $\sigma^N$ would be served in $N$. In this way, since $\mathsf{f}(p^N) = p$, the service costs in $N$ are the same as the service costs in $M$, and $\ell(p^N) \le \alpha\ell(p)$. Thus

$\mathrm{cost}_{\mathrm{OPT}_N}(\sigma^N) \leq \alpha \cdot \mathrm{cost}_{\mathrm{OPT}}(\sigma)$. Summarizing,

$$\mathbb{E}[\mathrm{cost}_{\mathcal{A}}(\sigma)] \leq \mathbb{E}[\mathrm{cost}_{\mathcal{A}_N}(\sigma^N)] \leq r \cdot \mathrm{cost}_{\mathrm{OPT}_N}(\sigma^N) + c \leq \alpha r \cdot \mathrm{cost}_{\mathrm{OPT}}(\sigma) + c. \qquad \square$$

COROLLARY 2.2. *There is an* $O(\log \Delta \log\log \Delta \cdot \log n \log\log n)$*-competitive randomized MTS algorithm for MTS defined on metric spaces with diameter* $\Delta$.

*Proof.* Apply Theorem 1.7 on the original metric and obtain an $O(\log \Delta \log\log \Delta)$-path embedding into an ultrametric of size $\Gamma(n) = \mathrm{poly}(n)$. This ultrametric has $O(\log \Gamma(n) \log\log \Gamma(n))$ competitive algorithm [16]. Now apply Proposition 2.1 to obtain the claim. $\square$

The GST [29] can be stated as follows: Given a graph $G = (V, E)$ on $n$ vertices with a weight function $c : E \to \mathbb{R}_+$, and subsets of the vertices $g_1, \ldots, g_k \subset V$ (called *groups*), the objective is to find a minimum weight subtree $T$ of $G$ that contains at least one vertex from each $g_i$, $i \in [k]$. Under certain standard complexity assumptions, this is hard to approximate by a factor better than $\max\{\log^{2-\varepsilon} k, \log^{2-\varepsilon} n\}$ [19]. The current best upper bound on the approximation factor is $O(\log^2 n \log k)$ [17, 15]. In [17], an $O(\log n \log k)$ approximation algorithm for tree metrics is given, and the general case is reduced to tree metrics using Theorem 1.3. Again, we show that it is actually sufficient to use multiembedding for this problem.

As a first step we observe that the problem can be easily cast in terms of metric spaces instead of graphs: Given a graph $G = (V, E)$ with weights $w : E \to \mathbb{R}_+$, let $M = (V, d)$ be the shortest path metric induced by $G$ and $w$ on $V$. A tree $T$ in $M$ can be transformed into a tree $\hat{T}$ in $G$ such that the total weight in $\hat{T}$ is not larger than the total weight in $T$, and $\hat{T}$ contains all the vertices in $T$. This is done by replacing each edge in $T$ by the shortest path between its endpoints in $G$ and taking a spanning tree of the resulting subgraph. It therefore suffices to solve GST on metric spaces.

PROPOSITION 2.3. *Assume that a metric space* $M$ *is* $\alpha$*-path embedded in a metric space* $N$. *Assume in addition that there is a [randomized] polynomial time* $r$*-approximation algorithm for any GST instance with* $k$ *groups defined on* $N$. *Then there exists a [randomized] polynomial time* $2\alpha r$*-approximation algorithm for any GST instance with* $k$ *groups defined on* $M$.

*Proof.* We construct an approximation algorithm $\mathcal{A}$ for the instance $\sigma = (M; g_1, \ldots, g_k)$ as follows. Denote by $\mathsf{f} : N \to M$ the $\alpha$-path embedding of $M$ in $N$. Consider the following instance of GST: $\sigma_N = (N; \mathsf{f}^{-1}(g_1), \ldots, \mathsf{f}^{-1}(g_k))$. Let $\mathcal{A}_N$ be an $r$-approximation algorithm for $\sigma_N$. Let $T_N = \mathcal{A}_N(\sigma_N)$ be the tree constructed by $A_N$. Denote by $\mathsf{f}(T_N)$ the image graph of $T_N$; i.e., if $T_N = (V_N, E_N)$, then $\mathsf{f}(T_N) = (\mathsf{f}(V_N), \{\mathsf{f}(u)\mathsf{f}(v) | \ uv \in E_N\})$. The graph $\mathsf{f}(T_N)$ is connected, and its weight is at most the weight of $T_N$. It also spans at least one representative from each group. Algorithm $\mathcal{A}$ returns a spanning tree of $\mathsf{f}(T_N)$. This tree is a feasible solution and it satisfies $\mathrm{cost}_{\mathcal{A}}(\sigma) \leq \mathrm{cost}_{\mathcal{A}_N}(\sigma_N)$.

Consider the tree $\mathrm{OPT}(\sigma)$, double each edge in $\mathrm{OPT}(\sigma)$, and take an Euler tour $p$ of this graph. There exists a path in $N$, $p_N$, as in the statement of Definition 1.6, such that $\mathsf{f}(p_N) = p$. The path $p_N$ is a connected graph and spans at least one representative from each group $\mathsf{f}^{-1}(g_j)$. As the weight of $p$ is twice the weight of $\mathrm{OPT}(\sigma)$, we have

$$\mathrm{cost}_{\mathrm{OPT}_N}(\sigma_N) \leq \ell(p_N) \leq \alpha \ell(p) \leq 2\alpha \mathrm{cost}_{\mathrm{OPT}}(\sigma).$$

Summarizing,

$$\mathbb{E}[\mathrm{cost}_{\mathcal{A}}(\sigma)] \leq \mathbb{E}[\mathrm{cost}_{\mathcal{A}_N}(\sigma_N)] \leq r \, \mathrm{cost}_{\mathrm{OPT}_N}(\sigma_N) \leq 2\alpha r \, \mathrm{cost}_{\mathrm{OPT}}(\sigma). \qquad \square$$

Corollary 2.4. *There exists a polynomial time $O(\log\Delta\log\log\Delta\log n\log k)$-approximation algorithm for GST on metric spaces with diameter $\Delta$.*

**3. Multiembedding into ultrametrics.** The following theorem is a restatement of Theorem 1.7 in a more general form.

Theorem 3.1. *Given any metric space $M=(V,d)$ on $|V|=n$ points and diameter $\Delta$, for any $t\in\mathbb{N}$, $M$ is $O(t\min\{\log\Delta,\log n\})$-path embedded into an efficiently constructible ultrametric of size $\Gamma\le n^{\beta}$, where $\beta=\min\{(\log n)^{1/t},[t\log(4\Delta)]^{2/t}\}$.*

*Proof.* The construction of the multiembedding is motivated by the construction of subspaces approximating ultrametric in [6, 8], but instead of deleting points, we *duplicate* them. We then prove the bounds on the path distortion.

Let $\Delta$ be the diameter of $M$. Let $x$ and $\bar{x}$ be two points realizing the diameter of $M$, and assume without loss of generality that $|\{y\in M:\ d(x,y)<\Delta/4\}|\le n/2$ (otherwise, switch the roles of $x$ and $\bar{x}$). Define a series of sets $A_0=\{x\}$, and for $i\in\{1,2,\dots,t\}$, $A_i=\{y\in M|\ d(x,y)<i\Delta/4t\}$, and "shells" $S_i=A_i\setminus A_{i-1}$. Let $|V|=n$ and let $\varepsilon_i=|A_i|/n$.

The algorithm for constructing the multiembedding works as follows: Choose a shell $S_i$, $i\in[t]$. Recursively, construct a multiembedding of the subspace $A_i$ into an ultrametric $T_1$ and a multiembedding of the subspace $V\setminus A_{i-1}$ into an ultrametric $T_2$. To construct the multiembedding for $M$, we construct an ultrametric $T$ with root labelled with $\Delta$ and two children, one being $T_1$ and the other $T_2$. This is a multiembedding since the points in $S_i$ are essentially being "duplicated" at this stage. Note that this is a noncontractive multiembedding.

Next we prove an upper bound on the size of the resulting ultrametric $T$, assuming that the shell was chosen carefully enough. The bound we prove is $n^{\beta}$, where $\beta=\min\{(\log n)^{1/t},[t\log(4\Delta)]^{2/t}\}$.

We begin with the first bound. Let $\beta=\beta(n)=(\log n)^{1/t}$. The proof proceeds by induction on $n$ (whereas $t$ is fixed). There must exist $i\in[t]$ such that $\varepsilon_{i-1}\ge\varepsilon_i^{\beta}$. Indeed, note that $n^{-1}\le\varepsilon_0\le\varepsilon_{t+1}\le 1/2$. Assume on the contrary that $\varepsilon_{i-1}<\varepsilon_i^{\beta}$ for all $i\in[t]$; then

$$\varepsilon_0<\varepsilon_1^{\beta}<\cdots<\varepsilon_t^{\beta^t}\le\left(\frac{1}{2}\right)^{\log n}=\frac{1}{n},$$

which is a contradiction. Therefore we can fix $i$ such that $\varepsilon_{i-1}\ge\varepsilon_i^{\beta}$. Inductively, assume that the recursive process results in at most $(\varepsilon_i n)^{\beta(\varepsilon_i n)}\le(\varepsilon_i n)^{\beta}$ leaves in $T_1$ and at most $((1-\varepsilon_{i-1})n)^{\beta((1-\varepsilon_{i-1})n)}\le((1-\varepsilon_{i-1})n)^{\beta}$ leaves in $T_2$. Thus $|T|\le(\varepsilon_i^{\beta}+(1-\varepsilon_{i-1})^{\beta})n^{\beta}$. Since $\varepsilon_{i-1}\ge\varepsilon_i^{\beta}$, we have $\varepsilon_i^{\beta}+(1-\varepsilon_{i-1})^{\beta}\le\varepsilon_{i-1}+(1-\varepsilon_{i-1})=1$, and we are done.

We next prove the second bound. Let $\beta=\beta(\Delta)=[t\log(4\Delta)]^{2/t}$. The proof is by induction on (the rounded value of) $\Delta$. We claim that

$$\exists i\in[t]\quad\text{such that}\quad\varepsilon_{i-1}\ge\varepsilon_i^{\beta(\Delta/2)}n^{\beta(\Delta/2)-\beta(\Delta)}. \tag{3.1}$$

Indeed, assume on the contrary that no such $i$ exists. Set $a=\log(2\Delta)\ge 1$, so that $\beta(\Delta/2)=(ta)^{2/t}$ and $\beta(\Delta)=[t(a+1)]^{2/t}$. Denote $b=n^{(ta)^{2/t}-[t(a+1)]^{2/t}}$ and $c=(ta)^{2/t}$. The opposite of (3.1) then becomes $\varepsilon_{i-1}<\varepsilon_i^{c}b$ for any $i\in[t]$. Iterating this $t$ times, we get

$$\frac{1}{n}=\varepsilon_0<\varepsilon_t^{c^t}b^{1+c+c^2+\cdots+c^{t-1}}\le\varepsilon_t^{c^t}b^{c^{t-1}}\le b^{c^{t-1}}.$$

Thus

$$n^{(ta)^{2-2/t}\left[[t(a+1)]^{2/t}-(ta)^{2/t}\right]} < n,$$

but this is a contradiction, since an application of the mean value theorem implies the existence of $\xi \in [a, a+1]$, for which

$$(ta)^{2-2/t}\left[[t(a+1)]^{2/t}-(ta)^{2/t}\right]$$
$$= (ta)^{2-2/t}[2t^{-1+2/t}\xi^{-1+2/t}] = 2(ta)\left(\frac{a}{\xi}\right)^{1-2/t} \geq ta \geq 1.$$

Choose an index $i \in \{1, \ldots, t\}$ satisfying (3.1). Since $i \leq t$, $\Delta(A_i) \leq \Delta/2$. The choice of the index $i$, and using the inductive hypothesis, gives the required lower bound on the cardinality of $T$, since

$$|T| \leq (\varepsilon_i n)^{\beta(\Delta/2)} + [(1-\varepsilon_{i-1})n]^{\beta(\Delta)} \leq \varepsilon_{i-1} n^{\beta(\Delta)} + (1-\varepsilon_{i-1})n^{\beta(\Delta)} \leq n^{\beta(\Delta)}.$$

We note that the running time of the algorithm above is $O(n^2)$ on each vertex in the tree and therefore $O(n^{\beta+2})$ for the whole tree . A slight variation on this algorithm (and a more careful analysis) has an $O(n^{\max\{\beta,2\}})$ running time.

The multiembedding described above has the following properties:

1. The multiembedding is noncontractive.
2. The tree structure defining the ultrametric is a binary tree.[2]

Let $u$ be an internal vertex in the binary tree defining the ultrametric, and $T$ the subtree rooted at $u$. We can rename the subtrees rooted with the children of $u$ as $T_1$ and $T_2$ such that the following hold:

3. Let $x$ and $y$ be two points in $M$. If $\emptyset \neq \mathsf{f}^{-1}(x) \cap T \subset T_1$ and $\emptyset \neq \mathsf{f}^{-1}(y) \cap T \subset T_2$, then $d(x,y) \geq \Delta(T)/4t$.
4. $|\mathsf{f}(T_1)| \leq |\mathsf{f}(T)|/2$.
5. $\Delta(T_1) \leq \Delta(T)/2$.

We next show, using the properties above, that the path distortion of this multiembedding is at most $8t \log \min\{n, \Delta\}$. Let $p = \langle u_1, u_2, \ldots, u_m \rangle$ be a path in $M$ whose length is $\ell(p)$. We construct a path $\bar{p}$ on the leaves of $T$ whose length satisfies $\ell(\bar{p}) \leq 8t \log \min\{n, \Delta\} \cdot \ell(p)$. The proof proceeds by induction on the height of the tree defining the ultrametric.

We partition $p$ into subpaths as follows. Define a sequence of indices and a sequence of subtrees of the root: Let $j_1 = 1$, and let $\hat{T}_1 \in \{T_1, T_2\}$ be the subtree of the root that includes the longest prefix of $p$. Assume that inductively that we have already defined $j_{i-1}$ and $\hat{T}_{i-1} \in \{T_1, T_2\}$. Define $j_i$ to be the minimum index such that $u_{j_i}$ is the first point in $p$ after $u_{j_{i-1}}$ with no representative in $\hat{T}_{i-1}$. Let $\hat{T}_i \in \{T_1, T_2\}$ be the other subtree of the root. Assume that this process is finished with $j_s$, $\hat{T}_s$. Next we define another sequence of indexes: $k_s = m$; for $i < m$ we define $k_i$ to be the largest number, smaller than $j_{i+1}$, such that $u_{k_i}$ does not have a representative in $\hat{T}_{i+1}$. By the construction of $\hat{T}_i$, we have that $j_i \leq k_i$ and $u_{k_i}$ has a representative in $\hat{T}_i$. See Figure 3.1 for an example of such partition. We have partitioned $p$ into subpaths $(\langle u_{j_i}, \ldots, u_{k_i} \rangle)_i$ and $(\langle u_{k_i}, \ldots, u_{j_{i+1}} \rangle)_i$. Informally, a subpath $\langle u_{j_i}, \ldots, u_{k_i} \rangle$ will be realized in $\hat{T}_i$, while subpath $\langle u_{k_i}, \ldots, u_{j_{i+1}} \rangle$ will be realized in $T_1$.

[2]Note that, more generally, any ultrametric can be defined by a binary tree.

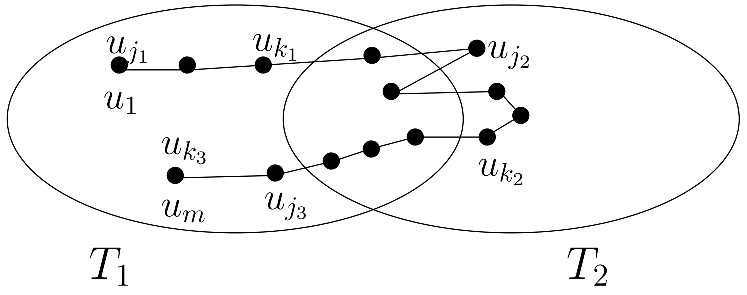


FIG. 3.1. *A partition of the path into subpaths.*

More formally, let $L = \ell(p)$, $L_{i_1,i_2} = \ell(\langle u_{i_1}, u_{i_1+1}, \dots, u_{i_2}\rangle)$, $n = |\mathsf{f}(T)|$, $n_1 = |\mathsf{f}(T_1)|$, $n_2 = |\mathsf{f}(T_2)|$, and $\Delta = \Delta(T)$, $\Delta_1 = \Delta(T_1)$, and $\Delta_2 = \Delta(T_2)$. We construct by induction on the tree structure $T$ a path $\bar{p}$ in $T$ whose length satisfies $\bar{L} = \ell(\bar{p}) \leq 8t \log\min\{\Delta, n\} \cdot L$.

By the induction hypothesis it is possible to construct, for any $i$, a path in $\hat{T}_i$ of representatives of $\langle u_{j_i}, \dots, u_{k_i}\rangle$ whose length is

$$\bar{L}_{j_i,k_i} \leq L_{j_i,k_i} \cdot 8t \log\min\{n, \Delta\}.$$

Next, for any $i$, we construct a path of representatives of $\langle u_{k_i}, \dots, u_{j_{i+1}}\rangle$. Note that $u_{k_i+1}, \dots, u_{j_{i+1}-1}$ have representatives in both $\hat{T}_i$ and $\hat{T}_{i+1}$. Therefore, we construct inductively a path from a representative of $u_{k_i+1}$ to a representative of $u_{j_{i+1}-1}$ in $T_1$, so $\bar{L}_{k_i,j_{i+1}} \leq L_{k_i,j_{i+1}} 8t \log\min\{n_1, \Delta_1\}$. We then connect the representative of $u_{k_i}$ with the representative of $u_{k_i+1}$ and the representative of $u_{j_{i+1}-1}$ with the representative of $u_{x_{i+1}}$; each such edge is of length at most the diameter of $T$, $\Delta$. We have therefore constructed a path of representatives of $\langle u_{k_i}, \dots, u_{j_{i+1}}\rangle$ whose length is $\bar{L}_{k_i,j_{i+1}} + 2\Delta$.

Since $u_{k_i}$ does not have a representative in $\hat{T}_{i+1}$ and $u_{j_{i+1}}$ does not have representative in $\hat{T}_i$, we conclude, using property 3 above, that $d_M(u_{k_i}, u_{j_{i+1}}) \geq \Delta/4t$, and so $\Delta \leq 4t \cdot L_{k_i,j_{i+1}}$. To summarize,

$$\begin{aligned}\bar{L}_{k_i,j_{i+1}} &\leq L_{k_i,j_{i+1}} 8t \log\min\{n_1, \Delta_1\} + 2\Delta \\ &\leq L_{k_i,j_{i+1}} 8t \big(\log\min\{n/2, \Delta/2\} + 1\big) \\ &= L_{k_i,j_{i+1}} 8t \log\min\{n, \Delta\}.\end{aligned}$$

We conclude

$$\begin{aligned}\bar{L} &= \sum_{i=1}^{s} \bar{L}_{j_i,k_i} + \sum_{i=1}^{s-1} \bar{L}_{k_i,j_{i+1}} \\ &\leq 8t \log\min\{n, \Delta\} \cdot \left(\sum_{i=1}^{s} L_{j_i,k_i} + \sum_{i=1}^{s-1} L_{k_i,j_{i+1}}\right) \\ &= 8t \log\min\{n, \Delta\} L. \qquad \square\end{aligned}$$

We end the discussion on multiembedding into ultrametrics with the following impossibility result.

PROPOSITION 3.2. *Consider the metric defined by a simple $N$-point path. Then any $\alpha$-path embedding of this metric in an ultrametric must have $\alpha = \Omega(\log n)$.*

*Proof.* Let $M = \{v_1, v_2, \ldots, v_n\}$ be the metric space on $n$ points such that $d_M(v_i, v_j) = |i - j|$. We prove that for any noncontractive multiembedding into an ultrametric $T$, any path of representatives of $\langle v_1, v_2, \ldots, v_n\rangle$ is of length at least $g(n) = \frac{n}{2}\log n$.

The proof proceeds by induction on $n$. For $n = 1$ the claim is trivial. For $n > 1$, let $\langle v'_1, v'_2, \ldots, v'_n\rangle$ be a path of representatives in $T$. Let $u = \mathrm{lca}_T(v'_1, v'_n)$, $\Delta(u) = d_T(v'_1, v'_n) \geq d_M(v_1, v_n) = n - 1$. Let $T_1$ be the subtree of the child of $u$ that contains $v'_1$. $T_1$ does not contain $v'_n$. Let $i_1 < n$ be the maximal $i$ such that $\{v'_1, \ldots, v'_{i_1}\} \subset T_1$. As $i_1 + 1$ is not contained in $T_1$, it must be that $d_T(v'_{i_1}, v'_{i_1+1}) \geq \Delta(u) \geq n - 1$. By the induction hypothesis,

$$\ell_T(\langle v'_1, \ldots, v'_{i_1}\rangle) \geq g(i_1), \qquad \ell_T(\langle v'_{i_1+1}, \ldots, v'_n\rangle) \geq g(n - i_1).$$

Since $g$ is a convex function, $(g(i_1) + g(n - i_1))/2 \geq g((i_1 + (n - i_1))/2) = g(n/2)$. We conclude

$$\begin{aligned}\ell_T(\langle v'_1, \ldots, v'_n\rangle) &\geq g(i_1) + (n-1) + g(n - i_1)\\ &\geq 2g\left(\frac{n}{2}\right) + n - 1 = 2\frac{n}{4}\log\frac{n}{2} + (n-1) \geq \frac{n}{2}\log n. \qquad \square\end{aligned}$$

**4. Multiembedding into trees.** In this section we consider multiembeddings into *arbitrary tree metrics*. We only have preliminary results. Specifically, we only consider two important types of metric spaces: expander graphs and the discrete cube with the Hamming metric, for which we obtain better results. For both of them the preceding sections proved an upper bound of $O(\log\log n \log\log\log n)$ and a lower bound of $\Omega(\log\log n)$ on the path distortion of multiembeddings into ultrametrics. (The lower bound follows since both metrics contain a path of length $\Omega(\log n)$.)

We begin with the observation that for $\Gamma = \infty$ it is easy to obtain 1-path embedding of any finite metric space into trees. This is achieved by defining an infinite tree metric as follows: joining all possible finite paths with a common root, where the first node in the path is connected with an edge weight of $\frac{\Delta}{2}$ to the root. Moreover, we have the following result.

Proposition 4.1. *Given a metric space $M$ defined by an unweighted graph of maximum degree $d$ and diameter $\Delta$, let $s \in \mathbb{N}$. Then $M$ can be $(2 + \frac{\Delta}{s})$-path embedded into a tree metric of size $nd^s$.*

*Proof.* Along the lines of the construction described above, we take all paths of length $s$ and join these with a common root, where the first node in the path is connected with an edge weight of $\frac{\Delta}{2}$ to the root. Obviously, there are at most $nd^s$ such paths. Notice that our choice of weights to the edges adjacent to the root guarantees that distances in the resulting tree are no smaller than the original distances. We next claim that the path distortion is at most $(2 + \frac{\Delta}{s})$. To see this, consider a path $p = \langle v_1, \ldots, v_\ell\rangle$ of length $\ell$. We partition $p$ into subpaths of length $s$: $p = p_1 p_2 \cdots p_t$, where $t = \lceil \ell/s \rceil$, $p_j = \langle v_{(j-1)s+1}, \ldots, v_{js}\rangle$ for $j < t$, and $p_t = \langle v_{(t-1)s+1}, \ldots, v_\ell\rangle$. Now the subpath $p_j$ is mapped to the appropriate path in the tree. Note that the length of the image path is $2\ell + (t-1)\Delta$. $\square$

This simple fact is particularly interesting for its implication for expander graphs. Let $G$ be an $(n, d, \gamma d)$ graph, i.e., a $d$-regular $n$-vertex graph whose second eigenvalue in absolute value is at most $\gamma d$. It is known [14] that such a graph has diameter at most $1 + \log_{1/\gamma} n$, and so we obtain the following.

Corollary 4.2. *Any $(n, d, \gamma d)$-graph has 3-path embedding in a tree of size $dn^{1+\log_{1/\gamma} d}$.*

We also note that, for the trees constructed in the proof of Proposition 4.1, it is particularly easy to obtain a better approximation algorithm for GST.

LEMMA 4.3. *Consider a tree metric $M = (V, d)$, where $V = P_1 \cup P_2 \cup \cdots \cup P_\ell$, $P_i = \langle v_1^i, v_2^i, \ldots, v_s^i \rangle$ is an unweighted simple path of length $s$, and $d(v_1^i, v_1^j) = \Delta$ for $i \neq j$. Then an instance of the GST with $k$ groups defined on $M$ has a $(1+\frac{2s}{\Delta})(1+\ln k)$ approximation algorithm.*

*Proof.* Consider a GST instance $g_1, \ldots, g_k$ defined on $M$. We first check whether there is a solution that is completely contained in one $P_i$. This can be checked in polynomial time by noting that if an optimal solution is contained in one $P_i$, then it is an interval. Thus we need to check only $\ell\binom{s+1}{2}$ intervals.

Otherwise, the optimal solution intersects $t > 1$ of the paths $P_1, \ldots, P_\ell$. Define a hitting set instance whose ground set is $\{P_1, \ldots, P_\ell\}$ and whose subsets are $g_1', \ldots, g_k'$, where $g_i' = \{P_j;\ P_j \cap g_i \neq \emptyset\}$. It follows that the optimal cost of the hitting set problem is at least $t-1$. The hitting set problem has a polynomial time $1+\ln k$-approximation algorithm [22, 25]. Let $S$ be the approximate solution for the hitting set. We define a solution for the GST instance by taking a natural path over $\cup\{P_i;\ P_i \in S\}$. Note that its length is at most $(\Delta + 2s)|S| \leq (\Delta + 2s)(t-1)(1 + \ln k)$. However, the cost of the optimal GST algorithm is at least $(t-1)\Delta$. $\square$

COROLLARY 4.4. *For fixed $d > \lambda$ there exist constants $c = c_{d,\lambda}$, $C = C_{d,\lambda}$, and polynomial $p(t) = p_{d,\lambda}(t)$ such that GST on $(n, d, \lambda)$ graphs has $p(n)$-time $(C \log k)$-approximation algorithm, and it is NP-hard to approximate within a factor of $c \log k$.*

*Proof.* The approximation algorithm follows from Proposition 4.1 and Lemma 4.3, by setting $s = \Delta$. The hardness result follows since an $(n, d, \lambda)$ graph contains a subset of $n^{\Omega_{d,\lambda}(1)}$ points that is $O_{d,\lambda}(1)$-approximated by an equilateral space [8]. GST on equilateral space is equivalent to a standard hitting set problem, which is NP-hard to approximate within a factor of $c \ln k$ [28, 26]. Usage of points not in this subspace ("Steiner points") can improve the approximation factor by at most a factor of two [29, 18]. $\square$

We next examine multiembedding of the $h$-dimensional hypercube with $n = 2^h$ vertices. Using Proposition 4.1 with $s = h/\log h$, and using $d = h$ and $\Delta = h$, we obtain $(\log\log n + 2)$-path embedding into trees of size $\Gamma(n) \leq n^2$. By applying Lemma 4.3 to that path-embedding, we obtain a polynomial time $O(\log k \log\log n)$-approximation algorithm to GST on the cube. Similarly to the expander graphs, it is hard to approximate instances of GST on the cube to within a $c \log k$ factor, for some constant $c > 0$, since the cube contains a subset of $n^{\Omega(1)}$ points that is $O(1)$-approximated by an equilateral space.

## 5. Discussion.

An interesting open problem is to determine worst case bounds for path distortion of multiembedding into trees of polynomial size. As indicated by the case of expander graphs, such bounds may be better than those for ultrametrics.

Results on multiembedding into trees directly reflect on the approximability of GST. As shown for expanders and hypercubes, it is possible that for special classes of metric spaces, a combination of improved path embedding and a specialized solution would yield (nearly) tight upper bounds. Our approach to showing the (near) tightness of the results in those cases stems from metric Ramsey-type considerations (i.e., the existence of large approximately equilateral subspace). Such considerations are, in fact, more general and may lead to more results of this type.[3]

Multiembedding into ultrametrics also implies multiembedding into $\ell_p^d$, where

[3]In [8] it is shown that any metric space contains a "large" subspace which is approximately an ultrametric, or a $k$-HST. Such trees were used in [19] to prove inapproximability results for GST. It is plausible that these techniques can be combined to obtain tight bounds for GST in specific metric

$d = O(\log n)$ with similar path distortion [9]. It is natural to ask whether better path distortion is possible for multiembedding into $\ell_1$ or $\ell_2$. Further study of multiembeddings in other settings and their applications seems an attractive direction for future research.

**Acknowledgments.** We thank Robi Krauthgamer and Assaf Naor for fruitful discussions.

REFERENCES

[1] N. Alon, R. M. Karp, D. Peleg, and D. West, *A graph-theoretic game and its application to the $k$-server problem*, SIAM J. Comput., 24 (1995), pp. 78–100.

[2] Y. Bartal, *Probabilistic approximations of metric space and its algorithmic application*, in Proceedings of the 37th Annual Symposium on Foundations of Computer Science, Burlington, VT, 1996, IEEE, Piscataway, NJ, pp. 183–193.

[3] Y. Bartal, *On approximating arbitrary metrics by tree metrics*, in Proceedings of the 30th Annual ACM Symposium on Theory of Computing, Dallas, TX, 1998, ACM, New York, pp. 183–193.

[4] Y. Bartal, *Graph decomposition lemmas and their role in metric embedding methods*, in Proceedings of the 12th Annual European Symposium on Algorithms, Bergen, Norway, 2004, Lecture Notes in Comput. Sci. 3221, Springer, New York, 2004, pp. 89–97.

[5] Y. Bartal, A. Blum, C. Burch, and A. Tomkins, *A* polylog($n$)*-competitive algorithm for metrical task systems*, in Proceedings of the 29th Annual ACM Symposium on Theory of Computing, El Paso, TX, 1997, ACM, New York, pp. 711–719.

[6] Y. Bartal, B. Bollobás, and M. Mendel, *A Ramsey-type theorem for metric spaces and its application for metrical task systems and related problems*, in Proceedings of the 42nd Annual Symposium on Foundations of Computer Science, Las Vegas, NV, 2001, IEEE, Piscataway, NJ, pp. 396–405.

[7] Y. Bartal, M. Charikar, and D. Raz, *Approximating* min-sum *$k$-clustering in metric spaces*, in Proceedings of the 33rd Annual ACM Symposium on Theory of Computing, Crete, Greece, 2001, ACM, New York, pp. 11–20.

[8] Y. Bartal, N. Linial, M. Mendel, and A. Naor, *On metric Ramsey-type phenomena*, in Proceedings of the 35th Annual ACM Symposium on Theory of Computing, San Diego, CA, 2003, ACM, New York, pp. 463–472.

[9] Y. Bartal, N. Linial, M. Mendel, and A. Naor, *Low dimensional embeddings of ultrametrics*, European J. Combin., 25 (2004), pp. 87–92.

[10] Y. Bartal and M. Mendel, *Multi-embedding and path approximation of metric spaces*, in Proceedings of the 14th Annual ACM-SIAM Symposium on Discrete Algorithms, Baltimore, MD, 2003, SIAM, Philadelphia, pp. 424–433.

[11] A. Borodin, N. Linial, and M. Saks, *An optimal online algorithm for metrical task systems*, J. ACM, 39 (1992), pp. 745–763.

[12] J. Bourgain, *On Lipschitz embedding of finite metric spaces in Hilbert space*, Israel J. Math., 52 (1985), pp. 46–52.

[13] M. Charikar, C. Chekuri, A. Goel, S. Guha, and S. Plotkin, *Approximating a finite metric by a small number of tree metrics*, in Proceedings of the 39th Annual IEEE Symposium on Foundations of Computer Science, Palo Alto, CA, 1998, IEEE, Piscataway, NJ, pp. 379–388.

[14] F. R. K. Chung, *Diameters and eigenvalues*, J. Amer. Math. Soc., 2 (1989), pp. 187–196.

[15] J. Fakcharoenphol, S. Rao, and K. Talwar, *A tight bound on approximating arbitrary metrics by tree metrics*, in Proceedings of the 35th Annual ACM Symposium on Theory of Computing, San Diego, CA, 2003, ACM, New York, pp. 448–455.

[16] A. Fiat and M. Mendel, *Better algorithms for unfair metrical task systems and applications*, SIAM J. Comput., 32 (2003), pp. 1403–1422.

[17] N. Garg, G. Konjevod, and R. Ravi, *A polylogarithmic approximation algorithm for the group Steiner tree problem*, J. Algorithms, 37 (2000), pp. 66–84.

[18] A. Gupta, *Steiner points in tree metrics don't (really) help*, in Proceedings of the 12th Annual ACM-SIAM Symposium on Discrete Algorithms, Washington, DC, 2001, SIAM, Philadelphia, pp. 220–227.

spaces.

[19] E. Halperin and R. Krauthgamer, *Polylogarithmic inapproximability*, in Proceedings of the 35th Annual ACM Symposium on Theory of Computing, San Diego, CA, 2003, ACM, New York, pp. 585–594.

[20] P. Indyk, *Algorithmic applications of low-distortion geometric embeddings*, in Proceedings of the 42nd Annual Symposium on Foundations of Computer Science, Las Vegas, NV, 2001, IEEE, Piscataway, NJ, pp. 10–33.

[21] P. Indyk and J. Matoušek, *Low distortion embeddings of finite metric spaces*, in Handbook of Discrete and Computational Geometry, 2nd ed., CRC Press, Boca Raton, FL, 2004, Chapter 8.

[22] D. Johnson, *Approximation algorithms for combinatorial problems*, J. Comput. System Sci., 9 (1974), pp. 256–278.

[23] J. Kleinberg and E. Tardos, *Approximation algorithms for classification problems with pairwise relationships: Metric labeling and Markov random fields*, J. ACM, 49 (2002), pp. 616–639.

[24] N. Linial, E. London, and Y. Rabinovich, *The geometry of graphs and some of its algorithmic applications*, Combinatorica, 15 (1995), pp. 215–245.

[25] L. Lovász, *On the ratio of optimal integral and fractional covers*, Discrete Math., 13 (1975), pp. 383–390.

[26] C. Lund and M. Yannakakis, *On the hardness of approximating minimization problems*, J. ACM, 41 (1994), pp. 960–981.

[27] Y. Rabinovich and R. Raz, *Lower bounds on the distortion of embedding finite metric spaces in graphs*, Discrete Comput. Geom., 19 (1998), pp. 79–94.

[28] R. Raz and S. Safra, *A sub-constant error-probability low-degree test, and a sub-constant error-probability PCP characterization of NP*, in Proceedings of the 29th Annual ACM Symposium on Theory of Computing, El Paso, TX, 1997, ACM, New York, pp. 475–484.

[29] G. Reich and P. Widmayer, *Beyond Steiner's problem: A VLSI oriented generalization*, in Graph-Theoretic Concepts in Computer Science (Proceedings of the 15th International Workshop, WG '89), Lecture Notes in Comput. Sci. 411, Springer, New York, 1990, pp. 196–210.

SIAM J. COMPUT. © 2004 Society for Industrial and Applied Mathematics
Vol. 34, No. 2, pp. 261–276

# BOUNDED-DEPTH FREGE LOWER BOUNDS FOR WEAKER PIGEONHOLE PRINCIPLES[*]

JOSHUA BURESH-OPPENHEIM[†], PAUL BEAME[‡], TONIANN PITASSI[†], RAN RAZ[§], AND ASHISH SABHARWAL[‡]

**Abstract.** We prove a quasi-polynomial lower bound on the size of bounded-depth Frege proofs of the pigeonhole principle $PHP^m_n$ where $m = (1 + 1/\text{polylog } n)n$. This lower bound qualitatively matches the known quasi-polynomial-size bounded-depth Frege proofs for these principles. Our technique, which uses a switching lemma argument like other lower bounds for bounded-depth Frege proofs, is novel in that the tautology to which this switching lemma is applied remains random throughout the argument.

**Key words.** propositional proof complexity, pigeonhole principle

**AMS subject classifications.** 03F20, 68Q17, 68R10

**DOI.** 10.1137/S0097539703433146

## 1. Introduction.

The propositional pigeonhole principle asserts that $m$ pigeons cannot be placed in $n$ holes with at most one pigeon per hole whenever $m$ is larger than $n$. It is an exceptionally simple fact that underlies many theorems in mathematics and is the most extensively studied combinatorial principle in proof complexity. (See [24] for an excellent survey on the proof complexity of pigeonhole principles.) It can be formalized as a propositional formula, denoted $PHP^m_n$, in a standard way.

Proving superpolynomial lower bounds on the length of propositional proofs of the pigeonhole principle when $m = n+1$ has been a major achievement in proof complexity. The principle can be made weaker (and hence easier to prove) by increasing the number of pigeons relative to the number of holes, or by considering fewer of the possible mappings of pigeons to holes. Two well-studied examples of the latter weakenings, the onto pigeonhole principle and the functional pigeonhole principle, only rule out, respectively, surjective and functional mappings from pigeons to holes. In this paper, we will prove lower bounds that apply to all of these variations of the basic pigeonhole principle.

For all $m > n$, Buss [10] has given polynomial-size Frege proofs of $PHP^m_n$. He uses families of polynomial-size formulas that count the number of 1's in an $N$-bit string and Frege proofs of their properties to show that the number of pigeons successfully mapped injectively can be at most the number of holes.

In weaker proof systems, where such formulas cannot be represented, the proof complexity of the pigeonhole principle depends crucially on the number of pigeons, $m$, as a function of the number of holes, $n$. As $m$ increases, the principle becomes weaker (easier to prove) and in turn the proof complexity question becomes more

[*]Received by the editors August 13, 2003; accepted for publication (in revised form) May 21, 2004; published electronically December 1, 2004.
http://www.siam.org/journals/sicomp/34-2/43314.html

[†]Computer Science Department, University of Toronto, Toronto, ON M5S 3G4, Canada (bureshop@cs.toronto.edu, toni@cs.toronto.edu). The research of the third author was supported by U.S.–Israel BSF grant 98-00349.

[‡]Computer Science and Engineering, University of Washington, Seattle, WA 98195-2350 (beame@cs.washington.edu, ashish@cs.washington.edu). The research of the second author was supported by NSF grant CCR-0098066.

[§]Faculty of Math and Computer Science, Weizmann Institute of Science, Rehovot 76100, Israel (ranraz@wisdom.weizmann.ac.il).

difficult. We review the basics of what is known for resolution and bounded-depth Frege systems below. Generally, the weak pigeonhole principle has been used to refer to $PHP_n^m$ whenever $m$ is at least a constant factor larger than $n$. We will be primarily concerned with forms of the pigeonhole principle that are significantly weaker than the usual pigeonhole principle but somewhat stronger than these typical weak forms.

For the resolution proof system, the complexity of the pigeonhole principle is essentially resolved. In 1985, Haken proved the first superpolynomial lower bounds for unrestricted resolution proofs of $PHP_n^m$ for $m = n+1$ [13]. This lower bound was generalized by Buss and Turán [11] for $m < n^2$. For the next ten years, the resolution complexity of $PHP_n^m$ for $m \geq n^2$ was completely open. A recent result due to Raz [22] gives exponential resolution lower bounds for the weak pigeonhole principle, and subsequently Razborov resolved the problem for most interesting variants of the pigeonhole principle [25].

Substantially less is known about the complexity of the pigeonhole principle in bounded-depth Frege systems, although strong lower bounds are known when the number of pigeons $m$ is close to the number of holes $n$. Ajtai proved superpolynomial lower bounds for $PHP_n^{n+1}$ with an ingenious blend of combinatorics and nonstandard model theory [2, 3]. This result was improved to exponential lower bounds in [7, 21, 17]. It was observed in [5] that the above lower bounds can in fact be applied to $PHP_n^m$ for $m \leq n + n^{\epsilon}$ for some $\epsilon$ that falls off exponentially in the depth of the formulas involved in the proof.

For the case of larger $m$ (the topic of this paper), the complexity of bounded-depth Frege proofs of $PHP_n^m$ is slowly emerging, with surprising and interconnected results. There are several deep connections between the complexity of the weak pigeonhole principle and other important problems. First, lower bounds for bounded-depth Frege proofs of the weak pigeonhole principles suffice to show unprovability results for the $P$ versus $NP$ statement (see [24]). Second, the long-standing question of whether or not the existence of infinitely many primes has an $I\Delta_0$ proof is closely related to the complexity of weak pigeonhole principle in bounded-depth Frege systems [20]. Third, the question is closely related to the complexity of approximate counting [19].

In bounded-depth Frege systems more powerful than resolution, there are a few significant prior results concerning the proof complexity of weak pigeonhole principles: There are bounded-depth Frege proofs of $PHP_n^m$ for $m$ as small as $n + n/\text{polylog } n$ of quasi-polynomial size [20, 16, 18]; thus exponential lower bounds for the weak pigeonhole principle are out of the question. In fact, this upper bound is provable in a very restricted form of bounded-depth Frege where all lines in the proof are disjunctions of polylog $n$-sized conjunctions, a proof system known as $Res(\text{polylog } n)$. On the other hand, [26] shows exponential lower bounds for $PHP_n^{2n}$ in $Res(k)$, a proof system which allows lines to be disjunctions of size-$k$ conjunctions for $k$ almost $\sqrt{\log n}$.

In this paper we prove quasi-polynomial lower bounds for the weak pigeonhole principle whenever $m \leq n + n/\text{polylog } n$. More precisely, we show the following.

Main Result. *For any integers $a, h > 0$, there exists an integer $c$ such that any depth-$h$ proof of $PHP_n^m$, where $m \leq n + n/\log^c n$, requires size $2^{\log^a n}$.*

This is a substantial improvement over previous lower bounds. Furthermore, the quantification of $a$, $h$, and $c$ cannot be easily improved without running into the upper bound of [4]. Our proof technique applies a switching lemma to a weaker tautology based on certain bipartite graphs. This type of tautology was introduced in [9]. Although we rely heavily on the simplified switching lemma arguments presented in [6, 27], one major difference from previous switching-lemma-based proofs is that both the tautologies themselves and the restrictions we consider remain random throughout most of the argument.

## 2. Overview.

The high-level schema of our proof is not new. Ignoring parameters for a minute, we start with an alleged proof of $PHP^m_n$ of small size. We then show that assigning values to some of the variables in the proof leaves us with a sequence of formulas, each of which can be represented as a particular type of decision tree of small height. This part of the argument is generally referred to as the switching lemma. We then prove that the leaves of any such short tree corresponding to a formula in the proof must all be labelled 1 if the proof is to be sound. Finally, we show that the tree corresponding to $PHP^m_n$ has leaves labelled 0, which is a contradiction since it must appear as a formula in the alleged proof. We now overview the lower bound components in more detail.

The lower bounds for bounded-depth Frege proofs of $PHP^{n+1}_n$ [3, 7, 21, 17] used *restrictions*, partial assignments of values to input variables, and iteratively applied "switching lemmas" with respect to random choices of these restrictions. The first switching lemmas [12, 1, 14] showed that after one applies a randomly chosen restriction that assigns values to many, but far from all, of the input variables, with high probability one can convert an arbitrary DNF formula with small terms into a CNF formula with small clauses (hence the name). More generally, such switching lemmas allow one to convert arbitrary DNF formulas with small terms into small height decision trees (which implies the conversion to CNF formulas with small clauses). The basic idea is that for each level of the formulas/circuits, one proves that a randomly chosen restriction will succeed with positive probability for all subformulas/gates at that level. One then fixes such a restriction for that level and continues to the next level. To obtain a lower bound, one chooses a family of restrictions suited to the target of the analysis. In the case of $PHP^m_n$, the natural restrictions to consider correspond to partial matchings between pigeons and holes.

The form of the argument by which switching lemmas are proven generally depends on the property that the ratio of the probability that an input variable remains unassigned to the probability that it is set to 0 (respectively, to 1) is sufficiently less than 1. In the case of a random partial matching that contains $(1-p)n$ edges applied to the variables of $PHP^m_n$, there are $pn$ unmatched holes and at least $pm$ unmatched pigeons. Hence, the probability that any edge-variable remains unassigned (i.e., neither used nor ruled out by the partial matching) is at least $p^2$. However, the partial matching restrictions set less than a $1/m$ fraction of variables to 1. Thus the proofs required that $p^2n < p^2m < 1$ and thus $p < n^{-1/2}$. This compares with choices of $p = n^{-O(1/h)}$ for depth-$h$ circuit lower bounds in the best arguments for parity proven in [14]. Hence, the best-known lower bound on the size of depth-$h$ circuits computing parity is of the form $2^{n^{\Omega(1/h)}}$, while the best-known lower bound on the size of depth-$h$ proofs of $PHP^{n+1}_n$ is of the form $2^{n^{2^{-O(h)}}}$.

A problem with extending the lower bounds to $PHP^m_n$ for larger $m$ is that, after a partial matching restriction is applied, the absolute difference between the number of pigeons and holes does not change, but the number of holes is dramatically reduced. This can qualitatively change the ratio between pigeons and holes. If this is too large, then the probability that variables remain unassigned grows dramatically and, in the next level, the above argument does not work at all. For example, with the above argument, if the difference between the number of pigeons and holes is as large as $n^{3/4}$, then after only one round the above argument will fail. The extension in [5] to lower bound proofs for $PHP^{n+n^{\epsilon_h}}_n$ for formulas of depth $h$ relies on the fact that even after $h$ rounds of restrictions the gap is small enough that there is no such qualitative change; but this is the limit using the probabilities as above.

We are able to resolve the above difficulties for $m$ as large as $n + n/\text{polylog } n$. In particular, we increase the probability that variables are set to 1 to $1/\text{polylog } n$ from $1/m$ by restricting the matchings to be contained in bipartite graphs $G$ of polylog $n$ degree. Thus we can keep as many as $n/\text{polylog } n$ of the holes unmatched in each round. Therefore, by choosing the exponents in the polylog $n$ carefully as a function of the depth of the formulas, we can tolerate gaps between the number of pigeons and the number of holes that are also $n/\text{polylog } n$.

A difficulty with this outline is that one must be careful throughout the argument that the restrictions one chooses do not remove all the neighbors of a node without matching it, which would simplify the pigeonhole principle to a triviality. It is not at all clear how one could explicitly construct low-degree graphs such that some simple additional condition on the restrictions that we choose at each stage could enforce the desired property. It is unclear even how one might do this nonconstructively because it is not clear what property of the random graph would suffice.

Instead, unlike previous arguments, we do not fix the graph in advance; we keep the input graph random throughout the argument and consider for each such graph $G$ its associated proof of the pigeonhole principle restricted to $G$. Since we do not know what $G$ is at each stage, we cannot simply fix the restriction as we deal with each level; we must keep that random as well. Having done this, we can use simple Chernoff bounds to show that, for almost all combinations of graphs and restrictions, the degree at each level will not be much smaller than the expected degree, so the pigeonhole principle will remain far from trivial. We adjust parameters to reduce the probability that a restriction fails to simplify a given level so that it is much smaller than the number of levels. Then we apply the probabilistic method to the whole experiment involving the graph $G$ as well as the sequence of restrictions.

There is one other technical point that is important in the argument. In order for the probabilities in the switching lemma argument to work out, it is critical that the degrees of vertices in the graph after each level of restriction is applied are decreased significantly at each step as well as being small in the original graph $G$. Using another simple Chernoff bound we show that the degrees of vertices given almost all combinations of graphs and restrictions will not be much larger than their expected value, and this suffices to yield the decrease in degree.

Overall, our argument is expressed in much the same terms as those in [6, 27], although we find it simpler to omit formally defining $k$-evaluations as separate entities. One way of looking at our technique is that we apply two very different kinds of random restrictions to a proof of $PHP^m_n$: first, one that sets many variables to 0, corresponding to the restriction of the problem to the graph $G$, and then one that sets partial matchings for use with the switching lemma.

## 3. Frege proofs and $PHP(G)$.

A *formula* is a tree whose internal nodes are labelled by either $\vee$ (fanin 2) or $\neg$ (fanin 1) and whose leaves are labelled by variables. Given a node in this tree, the full tree rooted at that node is called a (not necessarily proper) *subformula* of the original formula. If a formula contains no connectives, then it has *depth* 0. Otherwise, the *depth* of a (sub)formula $A$ is the maximum number of alternations of connectives along any path from the root to leaf, plus one. The *merged form* of a formula $A$ is the tree such that all $\vee$'s labelling adjacent vertices of $A$ are identified into a single node of unbounded fanin, also labelled $\vee$.

A *Frege proof system* is specified by a finite set of sound and complete *inference* *rules*, rules for deriving new propositional formulas from existing ones by consistent substitution of formulas for variables in the rule. A typical example is the following, due to Schoenfield, in which $p, q, r$ are variables that stand for formulas and $p, q \vdash r$ denotes that $p$ and $q$ yield $r$ in one step:

Excluded middle: $\vdash \neg p \vee p$. Expansion rule: $p \vdash q \vee p$.

Contraction rule: $p \vee p \vdash p$. Associative rule: $p \vee (q \vee r) \vdash (p \vee q) \vee r$.

Cut rule: $p \vee q,\ \neg p \vee r \vdash q \vee r$.

We will say that the *size* of a Frege rule is the number of distinct subformulas mentioned in the rule. For example, the size of the cut rule above is 7; the subformulas mentioned are $p, q, r, \neg p, p \vee q, \neg p \vee r, q \vee r$.

DEFINITION 3.1. *A proof of a formula $A$ in Frege system $\mathcal{F}$ is a sequence of formulas $A_1, \dots, A_r = A$ such that $\vdash A_1$ and for all $i > 1$ there is some (possibly empty) subset $\mathcal{A} \subset \{A_1, \dots, A_{i-1}\}$ such that $\mathcal{A} \vdash A_i$ is a substitution instance of a rule of $\mathcal{F}$.*

In what follows, let $\mathcal{F}$ be any fixed Frege system whose rules have size bounded by $f$.

DEFINITION 3.2. *For an $\mathcal{F}$-proof $\Pi$, let $\mathrm{cl}(\Pi)$ denote the closure of the set of formulas in $\Pi$ under subformulas. The size of a Frege proof $\Pi$ is $|\mathrm{cl}(\Pi)|$, the total number of distinct subformulas that appear in the proof. The depth of a proof is the maximum depth of the formulas in the proof.*

Let $G = (V_1 \cup V_2, E)$ be a bipartite graph where $|V_2| = n$ and $|V_1| = m > n$. We use $L(G)$ to denote the language built from the set of propositional variables $\{X_e : e \in E\}$, the connectives $\{\vee, \neg\}$, and the constants 0 and 1.

The following is a formulation of the onto and functional weak pigeonhole principle on the graph $G$. Note that if $G$ is not the complete graph $K_{m,n}$, then this principle is weaker than the standard onto and functional weak pigeonhole principle.

DEFINITION 3.3. *$PHP(G)$ is the OR of the following four (merged forms of) formulas in $L(G)$. In general, $i, j, k$ represent vertices in $G$, and $\Gamma(i)$ represents the set of neighbors of $i$ in $G$.*

1. $\bigvee_{(e,e') \in I} \neg(\neg X_e \vee \neg X_{e'})$ *for* $I = \{(e, e') : e, e' \in E; e = \{i, k\}, e' = \{j, k\}; i, j \in V_1; i \neq j; k \in V_2\}$*: two different pigeons go to the same hole.*
2. $\bigvee_{(e,e') \in I} \neg(\neg X_e \vee \neg X_{e'})$ *for* $I = \{(e, e') : e, e' \in E; e = \{k, i\}, e' = \{k, j\}; i, j \in V_2; i \neq j; k \in V_1\}$*: one pigeon goes to two different holes.*
3. $\bigvee_{i \in V_1} \neg \bigvee_{j \in \Gamma(i)} X_{\{i,j\}}$*: some pigeon has no hole.*
4. $\bigvee_{j \in V_2} \neg \bigvee_{i \in \Gamma(j)} X_{\{i,j\}}$*: some hole remains empty.*

*In fact, we consider an arbitrary orientation of the above formula whereby each $\vee$ is binary.*

## 4. Representing matchings by trees.

In this section we make minor modifications to standard definitions from [6, 27] to apply to the edge-variables given by bipartite graphs and not just complete bipartite graphs.

Let $G$ be a bipartite graph as in the last section and let $D$ denote the set of Boolean variables $X_e$ in $L(G)$. Assume there is an ordering on the nodes of $G$.

DEFINITION 4.1. *Two edges of $G$ are said to be inconsistent if they share exactly one endpoint. Two partial matchings $\rho_1, \rho_2$ on the graph $G$ are said to be consistent if no edge in $\rho_1$ is inconsistent with an edge in $\rho_2$. For a partial matching $\rho$, let $\mathrm{Im}(\rho)$ denote the set of nodes of $V_2$ that are matched by $\rho$.*

Definition 4.2. *For $\rho$ a partial matching on the graph $G$ that matches nodes $V_1' \subset V_1$ to nodes $V_2' \subset V_2$, we define $G|_\rho$ as the bipartite graph $((V_1 \setminus V_1') \cup (V_2 \setminus V_2'), E - (V_1' \times V_2 \cup V_1 \times V_2'))$.*

Definition 4.3. *A matching decision tree $T$ for $G$ is a tree where each internal node $u$ is labelled by a node of $G$, $v$, and each edge from a node $u$ is labelled by an edge of $G$ that touches $v$. Furthermore, given any path in the tree from the root to a node $u$, the labels of the edges along the path constitute a partial matching on $G$, called $path(u)$. Let $path(T) = \{path(u) : u \text{ is a leaf of } T\}$. If $v$ is a node of $G$ that appears as a label of some node in $T$, then $T$ is said to mention $v$.*

*Furthermore, each leaf of $T$ is labelled by $0$ or $1$ (if a tree satisfies the above conditions but its leaves remain unlabelled, we will call it a leaf-unlabelled matching decision tree). Let $T^c$ be the same as $T$ except with the value of each leaf-label flipped. If $U$ is the set of leaves of $T$ labelled $1$, let $disj(T)$ be the DNF formula $\bigvee_{u \in U} \bigwedge_{e \in path(u)} X_e$.*

Definition 4.4. *A complete (leaf-unlabelled) matching decision tree for $G$ is one in which, for each internal node $u$ labelled $v$ and each neighbor $v'$ of $v$ in $G|_{path(u)}$, there is an outgoing edge from $u$ labelled by $v'$.*

Definition 4.5. *Let $K$ be a subset of the nodes in $G$. The full matching tree for $K$ over $G$ is a leaf-unlabelled matching decision tree for $G$ defined inductively: If $K = \{k\}$, then the root of the tree is labelled by $k$, and for each edge $e$ in $G$ that touches $k$, there is an edge from the root of the tree labelled $e$. If there are no such edges, then we say that the full matching tree is empty.*

*If $K$ contains more than one node, let $k$ be its first node under the ordering and assume we have a full matching tree for $k$ called $T$. If $T$ is empty, then the entire tree is empty. Otherwise, at each leaf $u$ of $T$, attach the full matching tree for $K \setminus \{k\}$ over $G|_{path(u)}$. If this tree is empty, then remove the leaf $u$.*

Note that the full matching tree for any subset $K$ is complete. If the degree of each node in $K$ is at least $|K|$, then the full matching tree for $K$ is guaranteed to mention all nodes in $K$. Otherwise, it might not.

Lemma 4.6. *Let $T$ be a complete matching tree for $G$, and let $\rho$ be any partial matching on $G$. Let $d$ be the minimal degree of any node in $G$ mentioned by $T$. If $d > |\rho| + height(T)$, then there is a matching in $path(T)$ that is consistent with $\rho$.*

*Proof.* Assume we have found an internal node $u$ in $T$ labelled by $v$ in $G$ such that $path(u)$ is consistent with $\rho$. We will find a child $u'$ of $u$ such that $path(u')$ is still consistent with $\rho$. If $\rho$ includes the edge $\{v, v'\}$ for some $v'$, then there must be a $u'$ that matches $v$ with $v'$ in $T$, since $path(u)$ is consistent with $\rho$ (so $v'$ isn't already matched by $path(u)$) and since $T$ is complete. If not, then there must be a neighbor $v'$ of $v$ in $G$ that remains unmatched by $\rho$ and $path(u)$ because $d > |\rho| + height(T)$. Again, since $T$ is complete, there is a node $u'$ that matches $v$ with $v'$. ◻

Definition 4.7. *We call $F$ a matching disjunction if it is one of the constants $0$ or $1$, or it is a DNF formula with no negations over the variables $D$ such that the edges of $G$ corresponding to the variables in any one term constitute a partial matching. In the latter case, order the terms lexicographically based on the nodes they touch and the order of the nodes in $G$.*

Definition 4.8. *For $F$ a matching disjunction, the restriction $F|_\rho$ for $\rho$ a partial matching is another matching disjunction generated from $F$ as follows: Set any variable in $F$ corresponding to an edge of $\rho$ to $1$ and set any variable corresponding to an edge not in $\rho$ but incident to one of $\rho$'s nodes to $0$. If a variable in term $t$ is set to $0$, remove $t$ from $F$. Otherwise, if a variable in term $t$ is set to $1$, remove that variable from $t$.*

The DNF $disj(T)$ for a matching decision tree $T$ is always a matching disjunction.

DEFINITION 4.9. *A matching decision tree $T$ is said to represent a matching disjunction $F$ if, for every leaf $l$ of $T$, $F|_{path(l)} \equiv 1$ when $l$ is labelled 1 and $F|_{path(l)} \equiv 0$ when $l$ is labelled 0.*

A matching decision tree $T$ always represents $disj(T)$. Furthermore, if $\rho$ extends some matching $path(l)$ for $l$ a leaf of $T$, then $disj(T)|_\rho \equiv 0$ (1, respectively) if $l$ is labelled 0 (1).

DEFINITION 4.10. *Let $F$ be a matching disjunction. We define a tree $Tree_G(F)$ called the canonical decision tree for $F$ over $G$: If $F$ is constant, then $Tree_G(F)$ is one node labelled by that constant. Otherwise, let $C$ be the first term of $F$. Let $K$ be the nodes of $G$ touched by variables in $C$. The top of $Tree_G(F)$ is the full matching tree on $K$ over $G$. We replace each leaf $u$ of that tree with the tree $Tree_{G|_{path(u)}}(F|_{path(u)})$.*

The tree $Tree_G(F)$ will have all of its leaves labelled. It is designed to represent $F$ and to be complete.

DEFINITION 4.11. *For $T$ a matching decision tree and $\rho$ a matching, $T$ restricted by $\rho$, written $T|_\rho$, is a matching decision tree obtained from $T$ by first removing all edges of $T$ that are inconsistent with $\rho$ and retaining only those nodes of $T$ that remain connected to the root of $T$. Each remaining edge that corresponds to an element of $\rho$ is then contracted (its endpoints are identified and labelled by the label of the lower endpoint).*

LEMMA 4.12 (see [27, Lemma 4.8]). *For $T$ a matching decision tree and $\rho$ a matching,*

(a) *$disj(T)|_\rho \equiv disj(T|_\rho)$;*

(b) *$(T|_\rho)^c = T^c|_\rho$;*

(c) *if $T$ represents a matching disjunction $F$, then $T|_\rho$ represents $F|_\rho$.*

## 5. The lower bound.

Let $m = n+n/\log^c n$ for some integer $c > 0$, and let $h > 0$ be an integer (all log's are base 2). We generally assume that $n$ is large compared to the other parameters and that all subsequent expressions are integers. We will show that for any $a$ such that $8^h(a+3) < c$, any proof of $PHP^m_n = PHP(K_{m,n})$ of depth $h$ is of size greater than $2^{\log^a n}$. To do this we do not work directly with proofs of $PHP(K_{m,n})$ but rather with proofs of $PHP(G)$ for randomly chosen subgraphs $G$ of $K_{m,n}$.

More precisely, let $b = 8^h(a+3)$, define $d = \log^b n$, and observe that $a < b < c$.

Let $\mathcal{G}(m, n, d/n)$ be the uniform distribution on all bipartite graphs from $m$ nodes to $n$ nodes where each edge is present independently with probability $d/n$.

DEFINITION 5.1. *Let $H = (V_1 \cup V_2, E)$ be a fixed bipartite graph. Define $M^\ell(H)$ to be the set of all partial matchings of size $\ell$ in $H$, and for $I \subseteq V_2$ with $|I| = \ell$, let $M^\ell_I(H)$ be the set of all $\rho \in M^\ell(H)$ with $\mathrm{Im}(\rho) = I$. Define a partial distribution $\mathcal{M}^\ell(H)$ on $M^\ell(H)$ by first choosing a set $I \in V_2$ uniformly at random among all subsets of $V_2$ of size $\ell$, then choosing a $\rho \in M^\ell_I(H)$ uniformly at random; if $M^\ell_I(H)$ is empty, then no matching is chosen and the experiment fails.*

We now define several sequences of parameters for a probabilistic experiment. The meanings of these parameters will be explained after the definition of the experiment. For initial values, let

$$m_0 = m, \quad n_0 = n, \quad b_0 = b$$

and

$$k_0 = 7b_0/8, \quad \ell_0 = n_0 - n_0/\log^{k_0} n.$$

Then, for $1 \leq i \leq h$, we define recursively

$$m_i = m_{i-1} - \ell_{i-1}, \quad n_i = n_{i-1} - \ell_{i-1}, \quad b_i = b_{i-1} - k_{i-1}$$

and

$$k_i = 7b_i/8, \quad \ell_i = n_i - n_i/\log^{k_i} n.$$

In closed form,

$$n_i = n/(\log n)^{\sum_{j=0}^{i-1} k_j} = n/(\log n)^{b-b/8^i}, \quad m_i = n_i + (m-n), \quad b_i = b - \sum_{j=0}^{i-1} k_j = b/8^i$$

and

$$k_i = 7b/8^{i+1}, \quad \ell_i = (1 - 1/\log^{k_i} n)(n/(\log n)^{b-b/8^i}).$$

Now we are ready to define the experiment: Let $G_0 = G$ be a graph chosen randomly from the distribution $\mathcal{G}(m, n, d/n)$. For $0 \leq i \leq h-1$, let $\rho_i \sim \mathcal{M}^{\ell_i}(G_i)$ and define $G_{i+1} = G_i|_{\rho_i}$. (We say that the experiment fails during stage $i+1$ if the partial distribution $\mathcal{M}^{\ell_i}(G_i)$ fails to return an element $\rho_i$.) Observing that the choice of $\rho_i$ depends only on the edges of $G_i$ that are incident to $\mathrm{Im}(\rho_i)$, and that these are among the edges of $G_i$ that are removed to produce $G_{i+1}$, we have the following.

Proposition 5.2. *If this experiment succeeds up to stage $i$, then the distribution induced on $G_i$ is $\mathcal{G}(m_i, n_i, d/n)$.*

Thus, the expected degree of any pigeon in $G_i$ is $n_i d/n = \log^{b_i} n$. The expected degree of any hole in $G_i$ is $m_i d/n$, which is between $\log^{b_i} n$ and $2\log^{b_i} n$ since $n_i < m_i < 2n_i$ (because $c > b$). Let $\Delta_i \overset{def}{=} 6\log^{b_i} n$.

We make several observations about "bad" events in this experiment. Specifically, we bound the probability that any of the following fail:

$E_i$: The experiment succeeds up to stage $i$.

$A_i$: Every node in $G_i$ has degree at most $\Delta_i$.

$B_i$: Every node in $G_i$ has degree at least $(1/2)\log^{b_i} n$.

Lemma 5.3. *For $0 \leq i \leq h$, the probability, given $E_i$, of $\neg A_i$ is at most $(m_i + n_i)2^{-\log^{b_i} n} < 2^{-\log^{b_i - 1} n}$.*

*Proof.* The expected degree of any hole at stage $i$ will be $m_i d/n$. The expected degree of any pigeon will be $n_i d/n$. By the conditions on $m$ and $n$, the former quantity is at most twice the latter, which is equal to $\log^{b_i} n$. Fix a node in the graph and let $X$ be the random variable that represents its degree. This is a sum of Bernoulli trials since the edges occur independently with the same probability. Chernoff's bound tells us that $\mathbf{Pr}(X > 3\mu(X)) < (e^2/27)^{\mu(X)} < (1/2)^{\mu(X)}$. We know that $\log^{b_i} n \leq \mu(X) \leq 2\log^{b_i} n$ and $b_i \geq 2$, so we have the bound. ∎

Lemma 5.4. *For $0 \leq i \leq h$, the probability, given $E_i$, of $\neg B_i$ is at most $(m_i + n_i)2^{-\frac{1}{16}\log^{b_i} n} < 2^{-\log^{b_i - 1} n}$.*

*Proof.* The expected degree of any particular node in $G_i$ is at least $\log^{b_i} n$. Applying a Chernoff bound in the form $\mathbf{Pr}(X < \frac{1}{2}\mu(X)) < \exp(-\frac{1}{8}\mu(X))$, we have the result. ∎

Lemma 5.5. *For $0 \leq i \leq h-1$, the probability, given $E_i$, of $\neg E_{i+1}$ is at most $2^{-\log^{b_i - 1} n}$.*

*Proof.* This is less than the probability that a random graph from $\mathcal{G}(m_i, n_i, d/n)$ does not contain a perfect matching, which by Hall's theorem is less than the probability that there is a proper subset $S$ of the holes that has at least $n_i - |S|$ nonneighbors among the first $n_i$ pigeons. This is at most

$$\begin{aligned}\sum_{j=1}^{n_i-1} \binom{n_i}{j}^2 (1-d/n)^{j(n_i-j)} &\le \sum_{1\le j\le n_i/2} \binom{n_i}{j}^2 (1-d/n)^{j(n_i-j)} \\ &\quad + \sum_{n_i/2\le j\le n_i-1} \binom{n_i}{n_i-j}^2 (1-d/n)^{j(n_i-j)} \\ &\le 2 \sum_{1\le j\le n_i/2} [n_i^2(1-d/n)^{n_i-j}]^j \\ &\le 2 \sum_{1\le j\le n_i/2} [n_i^2 e^{-d(n_i-j)/n}]^j \\ &\le 2 \sum_{1\le j\le n_i/2} [n_i^2 e^{-dn_i/(2n)}]^j.\end{aligned}$$

By construction $dn_i/(2n) = \frac{1}{2}\log^{b_i} n$ and $n_i \le n$, so the failure probability is at most $2^{-\log^{b_i-1} n}$. □

We now develop the switching lemma argument. The overall structure uses the simplified counting techniques of [23] and [6]; however, the statement and proof are both complicated by the need to use probabilistic properties of the formulas themselves as well as the relationship of those properties to the restrictions under consideration. We first need some definitions.

DEFINITION 5.6. *For a bipartite graph $H = (V_1 \cup V_2, E)$ and integers $\ell$ and $\Delta$, let $N^{\ell,\Delta}(H)$ be the set of all $\rho$ in $M^\ell(H)$ such that all nodes of $H|_\rho$ have degree at most $\Delta$. For a set $I \subseteq V_2$ with $|I| = \ell$, let $N_I^{\ell,\Delta}(H)$ be the set of elements $\rho \in N^{\ell,\Delta}(H)$ with* $\mathrm{Im}(\rho) = I$.

For a particular $i$, the set $N^{\ell_i,\Delta_{i+1}}(G_i)$ represents in some sense the usable or "good" portion of all the matchings in $M^{\ell_i}(G_i)$. We therefore define the following event:

$C_i$: $\frac{|N^{\ell_i,\Delta_{i+1}}(G_i)|}{|M^{\ell_i}(G_i)|} \ge 1 - 2^{-\log^{b_{i+1}-2} n}$. Here $i < h$.

LEMMA 5.7. *For $0 \le i < h$, the probability, given $E_{i+1}$, of $\neg C_i$ is at most $1/n$.*

*Proof.* Observe that the expectation of

$$\frac{|N^{\ell_i,\Delta_{i+1}}_{\mathrm{Im}(\rho_i)}(G_i)|}{|M^{\ell_i}_{\mathrm{Im}(\rho_i)}(G_i)|},$$

conditional on success up to stage $i+1$, is precisely the probability that $\rho_i \in N^{\ell_i,\Delta_{i+1}}(G_i)$, conditional on success up to stage $i+1$, which is $> 1 - 2^{-\log^{b_{i+1}-1} n}$ by Lemma 5.3. Now, since

$$\frac{|N^{\ell_i,\Delta_{i+1}}_{\mathrm{Im}(\rho_i)}(G_i)|}{|M^{\ell_i}_{\mathrm{Im}(\rho_i)}(G_i)|}$$

is bounded above by 1, we can apply Markov's inequality to yield that the probability, conditional on success up to stage $i+1$, that

$$\frac{|N^{\ell_i,\Delta_{i+1}}_{\mathrm{Im}(\rho_i)}(G_i)|}{|M^{\ell_i}_{\mathrm{Im}(\rho_i)}(G_i)|} \leq 1 - n \cdot 2^{-\log^{b_{i+1}-1} n}$$

is at most $1/n$. The result follows by observing that $n \cdot 2^{-\log^{b_{i+1}-1} n} \leq 2^{-\log^{b_{i+1}-2} n}$, which is less than 1 because $b_j \geq 3$ for all $j$. $\square$

We are now ready to state the switching lemma.

LEMMA 5.8 (switching lemma). *Let $i, s, r$ be any integers such that $0 \leq i < h$, $0 < s \leq \Delta_{i+1}/\log^3 n$, and $r > 0$. Suppose $E_{i+1}$ and $A_i$ hold. Let $F$ be any matching disjunction with conjunctions of size $\leq r$ over the edge-variables of $G_i$. The probability that $Tree_{G_{i+1}}(F|_{\rho_i})$ has height $\geq s$ conditioned on the events (1) $\rho_i \in N^{\ell_i,\Delta_{i+1}}(G_i)$ and (2) $C_i$ is at most $2(720r/\log^{b_i/2} n)^{s/2}$.*

DEFINITION 5.9. *Let $stars(r,j)$ be the set of all sequences $\beta = (\beta_1, \dots, \beta_k)$ such that for each $i$, $\beta_i \in \{*, -\}^r \setminus \{-\}^r$ and the total number of $*$'s in $\beta$ is $j$.*

LEMMA 5.10 (see [6]). $|stars(r,j)| < (r/\ln 2)^j$.

LEMMA 5.11. *For $H$ a fixed bipartite graph with an ordering on its nodes, let $F$ be a matching disjunction with conjunctions of size $\leq r$ over the edge-variables of $H$, and let $S$ be the set of matchings $\rho \in N^{\ell,\Delta}(H)$ such that $Tree_{H|_\rho}(F|_\rho)$ has height $\geq s$. There is an injection from the set $S$ to the set*

$$\bigcup_{s/2 \leq j \leq s} M^{\ell+j}(H) \times stars(r,j) \times [\Delta]^s.$$

*Furthermore, the first component of the image of $\rho \in S$ is an extension of $\rho$.*

*Proof.* Let $F = C_1 \vee C_2 \vee \cdots$. If $\rho \in S$, then let $\pi$ be the partial matching labelling the first path in $Tree_{H|_\rho}(F|_\rho)$ of length $\geq s$ (actually, we consider only the first $s$ edges in $\pi$, starting from the root, and hence we assume $|\pi| = s$). Let $C_{\gamma_1}$ be the first term in $F$ not set to 0 by $\rho$, and let $K_1$ be the variables of $C_{\gamma_1}$ not set by $\rho$. Let $\sigma_1$ be the unique partial matching over $K_1$ that satisfies $C_{\gamma_1}|_\rho$, and let $\pi_1$ be the portion of $\pi$ that touches $K_1$.

Now define $\beta_1 \in \{*, -\}^r \setminus \{-\}^r$, so that the $p$th component of $\beta_1$ is an $*$ if and only if the $p$th variable in $C_{\gamma_1}$ is set by $\sigma_1$.

Continue this process to define $\pi_i$, $\sigma_i$, $K_i$, etc. (replacing $\rho$ with $\rho\pi_1 \dots \pi_{i-1}$ and $\pi$ with $\pi \setminus \pi_1 \dots \pi_{i-1}$) until some stage $k$ when we've exhausted all of $\pi$. Let $\sigma$ be the matching $\sigma_1 \dots \sigma_k$, and let $\beta$ be the vector $(\beta_1, \dots, \beta_k)$. Let $j = |\sigma|$ be the number of edges in $\sigma$. Note that $s/2 \leq j \leq s$. Observe that $\beta \in stars(r,s)$ and that $\rho\sigma \in M^{\ell+j}(H)$ and is an extension of $\rho$.

We now encode the differences between all the corresponding $\pi_i$ and $\sigma_i$ pairs in a single vector $\delta$ consisting of $|\pi| = s$ components, each in $\{1, \dots, \Delta\}$. Let $u_1$ be the smallest numbered node in $K_1$ and suppose that $\pi$ (in particular $\pi_1$) matches $u_1$ with some node $v_1$. Then the first component of $\delta$ is the natural number $x$ such that $v_1$ is the $x$th neighbor (under the ordering of nodes) of $u_1$ in the graph $H|_{\rho\sigma_2\sigma_3\dots\sigma_k}$. More generally, until the mates of all nodes in $K_1$ under $\pi_1$ have been determined, we determine the $p$th component of $\delta$ by finding the smallest numbered node $u_p$ of $K_1 \setminus \{u_1, \dots, u_{p-1}, v_1, \dots, v_{p-1}\}$ and then find its mate $v_p$ under $\pi_1$ and encode the position $x$ of $v_p$ in the order of the neighbors of $u_p$ in $H|_{\rho\sigma_2\sigma_3\dots\sigma_k}$. Once $K_1$ (and thus $\pi_1$) has been exhausted, the next component is based on the mates of the smallest numbered nodes in $K_2$ under $\pi_2$, until that is exhausted, etc., where the ordering about each vertex when dealing with $K_i$ is with respect to the graph $H|_{\rho\sigma_{i+1}\sigma_{i+2}\dots\sigma_k}$.

Finally, we define the image of $\rho \in S$ under the injection to be $(\rho\sigma, \beta, \delta)$. To prove that this is indeed an injection, we show how to invert it: Given $\rho\sigma_1 \dots \sigma_k$, we can identify $\gamma_1$ as the index of the first term of $F$ that is not set to 0 by it. Then, using $\beta_1$, we can reconstruct $\sigma_1$ and $K_1$. Next, reading the components of $\delta$ and the graph $H|_{\rho\sigma_2\dots\sigma_k}$, until all of $K_1$ is matched, we can reconstruct $\pi_1$. Then we can derive $\rho\pi_1\sigma_2 \dots \sigma_k$.

At a general stage $i$ of the inversion, we will know $\pi_1, \dots, \pi_{i-1}$ and $\sigma_1, \dots, \sigma_{i-1}$ and $K_1, \dots, K_{i-1}$. We use $\rho\pi_1 \dots \pi_{i-1}\sigma_i \dots \sigma_k$ to identify $\gamma_i$ and, hence, $\sigma_i$ and $K_i$ (using $\beta$). Then we get $\pi_i$ from $\delta$, $K_i$, and $\rho\sigma_{i+1} \dots \sigma_k$. After $k$ stages, we know all of $\sigma$ and can recover $\rho$. □

*Proof of Lemma* 5.8. Let $R_i$ be the set of $\rho_i \in N^{\ell_i, \Delta_{i+1}}(G_i)$ such that $C_i$ holds. By Lemma 5.7, the total probability of $R_i$ under the distribution $\mathcal{M}^{\ell_i}(G_i)$ is at least $(1 - 1/n)(1 - 2^{-\log^{b_{i+1}-2} n}) \geq 1 - 2/n$.

By Lemma 5.11 with $H \leftarrow G_i$, $\ell \leftarrow \ell_i$, and $\Delta \leftarrow \Delta_{i+1}$, a *bad* $\rho_i \in R_i$, for which $Tree_{G_{i+1}}(F|_{\rho_i})$ has height at least $s$, can be mapped uniquely to a triple $(\rho_i', \beta, \delta) \in M^{\ell_i + j}(G_i) \times stars(r, j) \times [\Delta_{i+1}]^s$, where $\rho_i'$ extends $\rho_i$ for some integer $j \in [s/2, s]$. We compute the probability of all such $\rho_i \in R_i$ associated with a given $j$, sum up over $j$, and then divide by the probability of $R_i$ to get the probability of a bad restriction conditioned on $R_i$. For fixed $j$, we can bound the probability of all bad $\rho_i \in R_i$ by bounding the ratio of the probability of each such $\rho_i$ to the probability of its image, $(\rho_i', \beta, \delta)$.

Let $I = \mathrm{Im}(\rho_i)$ and $I' = \mathrm{Im}(\rho_i')$. By definition, $I \subset I'$. The ratio of the probability of $\rho_i$ under $\mathcal{M}^{\ell_i}(G_i)$ to that of $\rho_i'$ under $\mathcal{M}^{\ell_i + j}(G_i)$ is precisely

$$\frac{\binom{n_i}{\ell_i + j} |M_{I'}^{\ell_i + j}(G_i)|}{\binom{n_i}{\ell_i} |M_I^{\ell_i}(G_i)|}.$$

Now any matching $\tau' \in M_{I'}^{\ell_i + j}(G_i)$ is an extension of some unique matching $\tau \in M_I^{\ell_i}(G_i)$. If $\tau \in N_I^{\ell_i, \Delta_{i+1}}(G_i)$, then the degrees of all nodes in $G_i|_\tau$ are at most $\Delta_{i+1}$, and so there are at most $\Delta_{i+1}^j$ matchings $\tau' \in M_{I'}^{\ell_i + j}(G_i)$ extending $\tau$. If $\tau \notin N_I^{\ell_i, \Delta_{i+1}}(G_i)$, then the degrees of all nodes in $G_i|_\tau$ are at most $\Delta_i$ because that is true of $G_i$ itself by assumption. Therefore there are at most $\Delta_i^j$ extensions $\tau' \in M_{I'}^{\ell_i + j}(G_i)$ of $\tau$. Since $\rho_i \in R_i$, $|N_I^{\ell_i, \Delta_{i+1}}(G_i)| / |M_I^{\ell_i}(G_i)|$ is at least $1 - 2^{-\log^{b_{i+1}-2} n}$, so the probability ratio is at most

$$\frac{\binom{n_i}{\ell_i + j}}{\binom{n_i}{\ell_i}} [\Delta_{i+1}^j + 2^{-\log^{b_{i+1}-2} n} \Delta_i^j] \leq \left[1 + 2^{1 - \log^{b_{i+1}-2} n} \left(\frac{\Delta_i}{\Delta_{i+1}}\right)^j\right] \left(\frac{\Delta_{i+1}(n_i - \ell_i)}{\ell_i}\right)^j$$

$$< \left[1 + 2^{1 - \frac{\Delta_{i+1}}{6 \log^2 n}} (\log n)^{k_i s}\right] \left(\frac{\Delta_{i+1} n_i}{\ell_i \log^{k_i} n}\right)^j \tag{5.1}$$

$$< \left[1 + 2^{1 - \frac{\Delta_{i+1}}{6 \log^2 n}} (\log n)^{k_i \Delta_{i+1} / \log^3 n}\right] \left(\frac{\Delta_{i+1} n_i}{\ell_i \log^{k_i} n}\right)^j \tag{5.2}$$

$$< \left(\frac{2\Delta_{i+1}}{\log^{k_i} n}\right)^j \tag{5.3}$$

$$= \left(\frac{12 \log^{b_{i+1}} n}{\log^{k_i} n}\right)^j.$$

Inequalities (5.1) and (5.2) follow from $j \leq s \leq \Delta_{i+1}/\log^3 n$ and the definitions of $\Delta_i$ and $\Delta_{i+1}$. Inequality (5.3) follows since $12k_i \log\log n < \log n$ for $n$ sufficiently large and the fact that $n_i/\ell_i = 1/(1-1/\log^{k_i} n)$, which is close to 1. Therefore the total probability of bad $\rho_i \in R_i$ associated with a given $j$ is at most

$$(12\log^{b_{i+1}-k_i} n)^j \times (r/\ln 2)^j \times \Delta_{i+1}^s \leq (20r\log^{b_{i+1}-k_i} n)^j \times (6\log^{b_{i+1}} n)^s.$$

Thus the total probability in question is at most

$$(1-2/n)^{-1}(6\log^{b_{i+1}} n)^s \times \sum_{s/2\leq j\leq s} (20r\log^{b_{i+1}-k_i} n)^j.$$

Since $b_{i+1} = b_i - k_i$ and without loss of generality $20r\log^{b_i-2k_i} n < 1/3$ (otherwise the probability bound in the lemma statement is meaningless), this quantity is at most $2(720r\log^{3b_i-4k_i} n)^{s/2} \leq 2(720r/\log^{b_i/2} n)^{s/2}$ since $3b_i - 4k_i = -b_i/2$. ∎

The above switching lemma will be used to show that, with respect to most matching restrictions, a depth-$h$ formula $A$ over $G$ can be represented by a short decision tree. We build these decision trees inductively on the subformulas of $A$. The tricky part, then, is when we are considering a $\vee$-gate of $A$, all of whose children already have short decision trees. This is exactly where we need to apply a restriction in order to get a "switch." The following definition formalizes this inductive representation by decision trees.

Definition 5.12. *For any graph $G$, let $S_G$ be a set of formulas of depth at most $h$ that is closed under subformulas and defined over $G$. For $\rho = \rho_0 \ldots \rho_{h-1}$ a matching on $G$, we define, for every $0 \leq i < h$, $\mathcal{T}_{\rho_0\ldots\rho_i}$, a mapping from formulas with depth $\leq i+1$ in $S_G$ to matching decision trees. It is defined inductively as follows: For a variable $X_e$, $\mathcal{T}_{\rho_0}(X_e)$ is $Tree_G(X_e)|_{\rho_0}$. For $0 < i < h$, for all formulas $A$ of depth $< i+1$, $\mathcal{T}_{\rho_0\ldots\rho_i}(A)$ is $\mathcal{T}_{\rho_0\ldots\rho_{i-1}}(A)|_{\rho_i}$. For $0 \leq i < h$, for all formulas $A$ of depth $i+1$, if $A = \neg B$, then $\mathcal{T}_{\rho_0\ldots\rho_i}(A)$ is $(\mathcal{T}_{\rho_0\ldots\rho_i}(B))^c$, and otherwise, if the merged form of $A$ is $\bigvee_{j\in J} B_j$, let $F$ be the matching disjunction $\bigvee_{j\in J} disj(\mathcal{T}_{\rho_0\ldots\rho_{i-1}}(B_j))$ and let $\mathcal{T}_{\rho_0\ldots\rho_i}(A)$ be the canonical matching tree $Tree_{G_{i+1}}(F|_{\rho_i})$.*

From the definition of $\mathcal{T}_\rho$, we have that if $\neg A$ is a formula in $S_G$, then $\mathcal{T}_\rho(\neg A) = (\mathcal{T}_\rho(A))^c$. Also, by Lemma 4.12, if $\bigvee_{i\in I} A_i$ is the merged form of some formula $A$ in $S_G$, then $\mathcal{T}_\rho(A)$ represents $\bigvee_{i\in I} disj(\mathcal{T}_\rho(A_i))$.

We would like to bound the heights of the decision trees in the image of $\mathcal{T}_\rho$ with respect to our experiment. Accordingly, we define the following events ($A$ is a formula over the variables of $G$ and $S_G$ is a set of such formulas):

$D_i(A)$: $\mathcal{T}_{\rho_0\ldots\rho_{i-1}}(A)$ has height at most $\log^a n$ if $A$ has depth at most $i$. Here $i \geq 1$.

$D_i(S_G)$: $D_i(A)$ holds for all formulas $A$ in $S_G$. Again, $i \geq 1$.

Lemma 5.13. *Let $a$ and $h$ be positive integers. For each graph $G$, let $S_G$ be a set of formulas closed under subformulas defined on the variables of $G$ such that $|S_G| \leq 2^{\log^a n}$ and each formula $A \in S_G$ has depth at most $h$. There exists a choice of $G$ and $\rho = \rho_0, \ldots, \rho_{h-1}$ such that the following conditions hold:*

1. *$\mathcal{T}_\rho(A)$ has height at most $\log^a n$ for all $A \in S_G$, and*
2. *every node in $G|_\rho$ has degree at least $\log^{a+3} n$.*

*Proof.* We proceed using the probabilistic method and the experiment above. We need to show that $E_h \cap B_h \cap D_h(S_G)$ has nonzero probability.

Now by Lemma 5.5, $\mathbf{Pr}[\neg E_{i+1} \mid E_i] < 2^{-\log^{b_i-1} n}$; by Lemma 5.3, $\mathbf{Pr}[\neg A_i \mid E_i] < 2^{-\log^{b_i-1} n}$; and by Lemma 5.4, $\mathbf{Pr}[\neg B_i \mid E_i] < 2^{-\log^{b_i-1} n}$. Furthermore, by Lemma 5.7,

$\mathbf{Pr}[\neg C_i \mid E_{i+1}] \le 1/n$. Let $A \in S_G$ be of depth $i < h$ with the merged form of $A$ equal to $\bigvee_{j\in J} Q_j$, and let $F$ be the matching disjunction $\bigvee_{j\in J} disj(\mathcal{T}_{\rho_0\ldots\rho_{i-1}}(Q_j))$. Observing that $b_h = b/8^h = (a+3)$, by Lemma 5.8 applied to $F$ with $r = s = \log^a n \le \Delta_h/\log^3 n$, we have

$$\begin{aligned}\mathbf{Pr}[\neg D_{i+1}(A) \mid E_{i+1} \wedge A_i \wedge D_i \wedge A_{i+1} \wedge C_i] &\le 2(720/\log^{b_i/2-a} n)^{(\log^a n)/2}\\ &\le 2(720/\log^{b_{h-1}/2-a} n)^{(\log^a n)/2}\\ &\le 2(720/\log^{3a+3} n)^{(\log^a n)/2} < 2^{-\log^a n}/n.\end{aligned}$$

Therefore, $\mathbf{Pr}[\neg D_{i+1} \mid E_{i+1} \wedge A_i \wedge D_i \wedge A_{i+1} \wedge C_i] \le 1/n$ since each $S_G$ contains at most $2^{\log^a n}$ disjunctions of depth $i+1$.

Therefore the total probability that some $E_i$, $A_i$, $B_i$, $C_i$, or $D_i$ fails is at most

$$\begin{aligned}&\sum_{i=0}^{h-1}\mathbf{Pr}[\neg E_{i+1} \mid E_i] + \sum_{i=0}^{h}\mathbf{Pr}[\neg A_i \mid E_i] + \sum_{i=0}^{h}\mathbf{Pr}[\neg B_i \mid E_i] + \sum_{i=0}^{h-1}\mathbf{Pr}[\neg C_i \mid E_{i+1}]\\ &\quad + \mathbf{Pr}[\neg D_1 \mid E_1 \wedge A_0 \wedge A_1 \wedge C_0]\\ &\quad + \mathbf{Pr}[\neg D_2 \mid E_2 \wedge A_1 \wedge D_1 \wedge A_2 \wedge C_1] + \cdots\\ &\quad + \mathbf{Pr}[\neg D_h \mid E_h \wedge A_{h-1} \wedge D_{h-1} \wedge A_h \wedge C_{h-1}].\end{aligned}$$

In total there are $5h+2$ terms in this sum, each of which is at most $1/n$, and thus the whole probability is $< 1$. □

The following three lemmas are adapted from [27].

LEMMA 5.14. *For any $G$, $\rho = \rho_0 \ldots \rho_{h-1}$, let $\Pi_G$ be a depth-$h$ $\mathcal{F}$-proof of $PHP(G)$, and let $\mathcal{T}_\rho$ be the mapping associated with* cl$(\Pi)$. *Let $C$ be a line in $\Pi$, and let $\mathcal{A}$ be the immediate ancestors of $C$ (if there are any), so that $\mathcal{A} \vdash C$. Let $\mathcal{B}$ be the subformulas of $\mathcal{A}$ and $C$ mentioned in the application of the rule which derives $C$ from $\mathcal{A}$. Finally, let $\sigma$ be a matching which extends soundly some $\sigma_A \in path(\mathcal{T}_\rho(A))$ for each $A \in \mathcal{A}$, some $\sigma_B \in path(\mathcal{T}_\rho(B))$ for each $B \in \mathcal{B}$, and some $\sigma_C \in path(\mathcal{T}_\rho(C))$. If $disj(\mathcal{T}_\rho(A))|_\sigma \equiv 1$ for all $A \in \mathcal{A}$, then $disj(\mathcal{T}_\rho(C))|_\sigma \equiv 1$.*

*Proof.* Let $\Lambda = \mathcal{A} \cup \mathcal{B} \cup \{C\}$. First note the following facts, where $\alpha, \beta \in \Lambda$ and $D(\alpha)$ is an abbreviation for $disj(\mathcal{T}_\rho(\alpha))$:

- $D(\alpha)|_\sigma \equiv 0$ or $D(\alpha)|_\sigma \equiv 1$;
- if $\neg\alpha \in \Lambda$, then $D(\neg\alpha)|_\sigma \equiv 1$ if and only if $D(\alpha)|_\sigma \equiv 0$;
- if $(\alpha \vee \beta) \in \Lambda$, then $D(\alpha \vee \beta)|_\sigma \equiv 1$ if and only if $D(\alpha)|_\sigma \equiv 1$ or $D(\beta)|_\sigma \equiv 1$.

Now consider the rule $R$ used to derive $C$ formulated as in the examples from section 3. The application of $R$ substitutes subformulas $A_p, A_q, A_r, \ldots$ in $\Lambda$ for each of the atoms $p, q, r, \ldots$ in $R$, and there is a derived correspondence mapping subformulas $F$ appearing in $R$ to formulas $A_F \in \Lambda$. Define a function $\tau$ on the atoms of $R$ by $\tau(p) = D(A_p)|_\sigma$ for each such atom $p$. By the first property, $\tau$ is a truth assignment to these atoms. Furthermore, by the other two properties, the truth assignment $\tau$ extends to all subformulas $F$ in $R$ so that $\tau(F) = D(A_F)|_\sigma$. Since $R$ is sound, if $\tau$ satisfies all formulas in $\mathcal{A}$, it will satisfy $C$ and thus $D(C)|_\sigma \equiv 1$. □

LEMMA 5.15. *Let $a, h > 0$. For each $G$, assume that $\Pi_G$ is a proof in $\mathcal{F}$ of $PHP(G)$ of size at most $2^{\log^a n}$ and depth at most $h$. There exists a choice of $G$ and $\rho = \rho_0, \ldots, \rho_{h-1}$ such that, for any line $C$ in $\Pi$, all leaves of $\mathcal{T}_\rho(C)$ are labelled by* 1.

*Proof.* Let $\rho$ and $G$ be as defined in Lemma 5.13 applied with $S_G = cl(\Pi_G)$. We proceed by (complete) induction on the lines in the proof. Assume every leaf of $\mathcal{T}_\rho$ for any line preceding $C$ is labelled 1. Let $\mathcal{A}$, $\mathcal{B}$, $\Lambda$ be as in Lemma 5.14. For any leaf

$l$ of $\mathcal{T}_\rho(C)$, we use Lemma 4.6 to find $\sigma$ that extends $path(l)$ and extends a matching in each of the sets $path(\mathcal{T}_\rho(A))$ for all $A \in \mathcal{A}$ and $path(\mathcal{T}_\rho(B))$ for all $B \in \mathcal{B}$. This is possible since there are at most $f$ trees to consider, and by Lemma 5.13 the sum of their heights is at most $f \log^a n < \log^{a+1} n$, which is less than the degree of $G_h$.

By assumption, $disj(\mathcal{T}_\rho(A))|_\sigma \equiv 1$ for all $A$ in $\mathcal{A}$. Hence, by Lemma 5.14, $disj(\mathcal{T}_\rho(C))|_\sigma \equiv 1$, so $l$ must be labelled 1. □

LEMMA 5.16. *For any $G$ and any $\rho = \rho_0, \ldots, \rho_{h-1}$, all leaves of $\mathcal{T}_\rho(PHP(G))$ have label* 0.

*Proof.* It suffices to show that $\mathcal{T}_\rho$ applied to each of the following types of formulas has all leaves labelled 0:

1. $\neg(\neg X_e \vee \neg X_{e'})$ for $e, e' \in E; e = \{i, k\}, e' = \{j, k\}; i, j \in V_1; i \neq j; k \in V_2$.
2. $\neg(\neg X_e \vee \neg X_{e'})$ for $e, e' \in E; e = \{k, i\}, e' = \{k, j\}; i, j \in V_2; i \neq j; k \in V_1$.
3. $\neg \bigvee_{j\in\Gamma(i)} X_{\{i,j\}}$ for $i \in V_1$.
4. $\neg \bigvee_{i\in\Gamma(j)} X_{\{i,j\}}$ for $j \in V_2$.

In fact, we will show that $\mathcal{T}_\rho$ applied to the complement of each of these formulas has all leaves labelled 1.

For a formula of the first type, $T = \mathcal{T}_\rho(\neg X_e \vee \neg X_{e'})$ must represent $disj(\mathcal{T}_\rho(\neg X_e)) \vee disj(\mathcal{T}_\rho(\neg X_{e'}))$. If $\rho$ sets the value of either $X_e$ or $X_{e'}$, then it must set one of $\neg X_e$ or $\neg X_{e'}$ to 1, and thus all leaves of $\mathcal{T}_\rho(\neg X_e \vee \neg X_{e'})$ are certainly labelled 1. Otherwise, for $l$ a leaf of $T$, $path(l)$ cannot contain both $e$ and $e'$. Without loss of generality, it does not contain $e$. By Lemma 4.6 applied to graph $G_h$, we can find $\sigma$ that extends $path(l)$ and is an extension of some matching in $\mathcal{T}_\rho(\neg X_e)$. But then $disj(\mathcal{T}_\rho(\neg X_e))|_\sigma \equiv 1$, so $l$ must be labelled 1. The argument is the same for formulas of the second type.

For a formula of the third type, $T = \mathcal{T}_\rho(\bigvee_{j\in\Gamma(i)} X_{\{i,j\}})$ must represent $\bigvee_{j\in\Gamma(i)} disj(\mathcal{T}_\rho(X_{\{i,j\}}))$. Hence, if $\rho$ sets $X_{\{i,j\}}$ to 1 for some $j \in \Gamma(i)$, then all leaves of $T$ are certainly labelled 1. Otherwise, for a leaf $l$ of $T$, if $path(l)$ touches node $i$, then $\bigvee_{j\in\Gamma(i)} disj(\mathcal{T}_\rho(X_{\{i,j\}}))|_{path(l)} \equiv 1$. Finally, if $path(l)$ does not touch node $i$, extend it to $\sigma = path(l) \cup \{i, j\}$ for some $j$ such that $X_{\{i,j\}}$ is not set by $\rho$. Then $disj(\mathcal{T}_\rho(X_{\{i,j\}}))|_\sigma \equiv 1$, so $l$ is labelled 1. Formulas of the fourth type follow in the same way. □

THEOREM 5.17. *For any $a, h > 0$, there exists a $c$ such that there is a bipartite graph $G$ from $m = n + n/\log^c n$ pigeons to $n$ holes that has no depth-$h$, $2^{\log^a n}$-size $\mathcal{F}$-proof of $PHP(G)$*

*Proof.* Assume that for all such $G$, there is a proof $\Pi_G$ of the required depth and size. For the $G$ in Lemma 5.15 there exists a $\rho$ such that, for every line $A$ in $\Pi_G$, $\mathcal{T}_\rho(A)$ has all leaves labelled 1. But $\mathcal{T}_\rho(PHP(G))$ has all leaves labelled 0 by Lemma 5.16. If $\Pi_G$ is to be a proof of $PHP(G)$, then $PHP(G)$ must appear in $\Pi_G$, so we have a contradiction. □

COROLLARY 5.18. *For any $a, h > 0$, there exists a $c$ such that there is no depth-$h$, $2^{\log^a n}$-size $\mathcal{F}$-proof of $PHP = PHP(K_{m,n})$ from $m = n + n/\log^c n$ pigeons to $n$ holes.*

## 6. Open questions.

Among the many unresolved proof complexity questions regarding the pigeonhole principle (see [24]) the most important open problem is to resolve the complexity of the weak pigeonhole principle with $2n$ or more pigeons and $n$ holes. This would have many implications for the metamathematics of the $P$ versus $NP$ statement, the complexity of approximate counting, and the proof-theoretic strength underlying elementary number theory.

In the proof presented here, we derived a switching lemma using simple restrictions that limit the space of truth assignments to a subcube where certain variables are set to 0 or to 1. While this fails with $2n$ pigeons, a more general class of restrictions may suffice. Possible generalizations include the projections suggested in [28], which also allow identification of variables, or restrictions given by linear equations. Two important results [15, 8] for bounded-depth Frege systems already employ such generalized switching lemmas in cases where direct restrictions fail (although the latter use is implicit). Bounded-depth Frege reductions, such as those in [8], may also be useful for resolving the $2n$ to $n$ case.

A potentially simpler problem that still gets to the heart of the matter is to prove quasi-polynomial lower bounds for $Res(\text{polylog } n)$ proofs of the weak pigeonhole principle which would match the upper bounds in [18]. New techniques seem to be required, however: it is interesting to note that our technique does not suffice for proving a lower bound on $PHP_n^{2n}$ even in $Res(\log n)$ (i.e., when $m = 2n$, our experiment is not likely to succeed even in the first round). This is because a successful switch is predicated on the restricted graph's having low degree. In this case, it would require a degree so low that the decision tree argument could not be carried out.

**Acknowledgments.** We are very grateful to Alan Woods for suggesting this problem a while back and for the many valuable comments and insights that he shared with us. We also want to thank Mike Molloy for some helpful discussions. Finally, we are grateful to Miki Ajtai for hinting that there may just not be a polynomial-size bounded-depth proof.

## REFERENCES

[1] M. Ajtai, *$\Sigma_1^1$-formulae on finite structures*, Ann. Pure Appl. Logic, 24 (1983), pp. 1–48.

[2] M. Ajtai, *The complexity of the pigeonhole principle*, in Proceedings of the 29th Annual IEEE Symposium on Foundations of Computer Science, White Plains, NY, 1988, pp. 346–355.

[3] M. Ajtai, *The complexity of the pigeonhole principle*, Combinatorica, 14 (1994), pp. 417–433.

[4] A. Atserias, *Improved bounds on the weak pigeonhole principle and infinitely many primes from weaker axioms*, Theoret. Comput. Sci., 295 (2003), pp. 27–39.

[5] P. Beame and S. Riis, *More on the relative strength of counting principles*, in Proof Complexity and Feasible Arithmetics, P. Beame and S. Buss, eds., DIMACS Ser. Discrete Math. Theoret. Comput. Sci. 39, AMS, Providence, RI, 1998, pp. 13–35.

[6] P. W. Beame, *A Switching Lemma Primer*, Tech. Report UW-CSE-95–07–01, Department of Computer Science and Engineering, University of Washington, 1994.

[7] P. W. Beame, R. Impagliazzo, J. Krajíček, T. Pitassi, P. Pudlák, and A. Woods, *Exponential lower bounds for the pigeonhole principle*, in Proceedings of the 24th Annual ACM Symposium on Theory of Computing, Victoria, BC, Canada, 1992, pp. 200–220.

[8] E. Ben-Sasson, *Hard examples for bounded depth frege*, in Proceedings of the 34th Annual ACM Symposium on Theory of Computing, New York, 2002, pp. 563–572.

[9] E. Ben-Sasson and A. Wigderson, *Short proofs are narrow—resolution made simple*, in Proceedings of the 31st Annual ACM Symposium on Theory of Computing, Atlanta, GA, 1999, pp. 517–526.

[10] S. Buss, *Polynomial size proofs of the pigeonhole principle*, J. Symbolic Logic, 57 (1987), pp. 916–927.

[11] S. Buss and G. Turán, *Resolution proofs of generalized pigeonhole principles*, Theoret. Comput. Sci., 62 (1988), pp. 311–317.

[12] M. Furst, J. B. Saxe, and M. Sipser, *Parity, circuits, and the polynomial-time hierarchy*, Math. Systems Theory, 17 (1984), pp. 13–27.

[13] A. Haken, *The intractability of resolution*, Theoret. Comput. Sci., 39 (1985), pp. 297–305.

[14] J. Håstad, *Almost optimal lower bounds for small depth circuits*, in Proceedings of the 18th Annual ACM Symposium on Theory of Computing, Berkeley, CA, 1986, pp. 6–20.

[15] R. Impagliazzo and N. Segerlind, *Counting axioms do not polynomially simulate counting gates*, in Proceedings of the 42nd Annual IEEE Symposium on Foundations of Computer Science, Las Vegas, NV, 2001, pp. 200–209.

[16] J. Krajíček, *Bounded Arithmetic, Propositional Logic and Complexity Theory*, Cambridge University Press, Cambridge, UK, 1996.

[17] J. Krajíček, P. Pudlák, and A. Woods, *Exponential lower bounds to the size of bounded depth Frege proofs of the pigeonhole principle*, Random Structures Algorithms, 7 (1995), pp. 15–39.

[18] A. Maciel, T. Pitassi, and A. R. Woods, *A new proof of the weak pigeonhole principle*, in Proceedings of the 32nd Annual ACM Symposium on Theory of Computing, Portland, OR, 2000, pp. 368–377.

[19] J. Paris and A. Wilkie, *Counting problems in bounded arithmetic*, in Methods in Mathematical Logic, Lecture Notes in Math. 1130, Springer-Verlag, Berlin, 1985, pp. 317–340.

[20] J. Paris, A. J. Wilkie, and A. R. Woods, *Provability of the pigeonhole principle and the existence of infinitely many primes*, J. Symbolic Logic, 53 (1988), pp. 1235–1244.

[21] T. Pitassi, P. W. Beame, and R. Impagliazzo, *Exponential lower bounds for the pigeonhole principle*, Comput. Complexity, 3 (1993), pp. 97–140.

[22] R. Raz, *Resolution lower bounds for the weak pigeonhole principle*, in Proceedings of the 34th Annual ACM Symposium on Theory of Computing, 2002, pp. 553–562.

[23] A. A. Razborov, *Bounded arithmetic and lower bounds in Boolean complexity*, in Feasible Mathematics II, P. Clote and J. Remmel, eds., Birkhäuser Boston, Boston, MA, 1995, pp. 344–386.

[24] A. A. Razborov, *Proof complexity of pigeonhole principles*, in Proceedings of the 5th International Conference on Developments in Language Theory, Vienna, Austria, Lecture Notes in Comput. Sci. 2295, Springer-Verlag, Berlin, 2002, pp. 110–116.

[25] A. A. Razborov, *Resolution lower bounds for perfect matching principles*, in Proceedings of the 17th Annual IEEE Conference on Computational Complexity, Montreal, PQ, Canada, 2002, pp. 17–26.

[26] N. Segerlind, S. R. Buss, and R. Impagliazzo, *A switching lemma for small restrictions and lower bounds for k-DNF resolution*, in Proceedings of the 43rd Annual IEEE Symposium on Foundations of Computer Science, 2002, pp. 604–616.

[27] A. Urquhart and X. Fu, *Simplified lower bounds for propositional proofs*, Notre Dame J. Formal Logic, 37 (1996), pp. 523–544.

[28] L. G. Valiant, *Reducibility by algebraic projections*, Enseign. Math. II. Sér., 28 (1982), pp. 253–268. Also in Logic and Algorithmic. An International Symposium Held in Honor of Ernst Specker, Zürich, 1980, E. Engeler, H. Läuchli, and V. Strassen, eds., Monographie 30 de L'Enseignement Mathématique, Université de Genève, 1982, pp. 365–380.

SIAM J. COMPUT. © 2004 Society for Industrial and Applied Mathematics
Vol. 34, No. 2, pp. 277–298

# A COMPLETE CHARACTERIZATION OF THE ALGEBRAS OF MINIMAL BILINEAR COMPLEXITY[*]

MARKUS BLÄSER[†]

**Abstract.** Let $R(A)$ denote the rank (also called bilinear complexity) of a finite dimensional associative algebra $A$. A fundamental lower bound for $R(A)$ is the so-called Alder–Strassen bound $R(A) \geq 2\dim A - t$, where $t$ is the number of maximal twosided ideals of $A$. An algebra is called an algebra of minimal rank if the Alder–Strassen bound is tight, i.e., $R(A) = 2\dim A - t$.

As the main contribution of this work, we characterize all algebras of minimal rank over arbitrary fields. This finally solves an open problem in algebraic complexity theory; see, for instance, [V. Strassen, *Handbook of Theoretical Computer Science*, J. van Leeuwen, ed., Elsevier Science, New York, 1990, Vol. A, pp. 634–672, section 12, Problem 4] or [P. Bürgisser, M. Clausen, and M. A. Shokrollahi, *Algebraic Complexity Theory*, Springer, New York, 1997, Problem 17.5].

**Key words.** associative algebras, Alder–Strassen bound, bilinear complexity, rank

**AMS subject classifications.** 68Q17, 16Z05

**DOI.** 10.1137/S0097539703438277

## 1. Introduction.

One of the most important problems in algebraic complexity theory is the question about the costs of multiplication, say of matrices, triangular matrices, or polynomials (modulo a fixed polynomial). Let $A$ be a finite dimensional associative $k$-algebra with identity 1. By fixing a basis of $A$, say $v_1, \dots, v_N$, we can define a set of bilinear forms corresponding to the multiplication in $A$. If $v_\mu v_\nu = \sum_{\kappa=1}^N \alpha_{\mu,\nu}^{(\kappa)} v_\kappa$ for $1 \leq \mu, \nu \leq N$ with *structural constants* $\alpha_{\mu,\nu}^{(\kappa)} \in k$, then these constants and the equation

$$\left(\sum_{\mu=1}^N X_\mu v_\mu\right)\left(\sum_{\nu=1}^N Y_\nu v_\nu\right) = \sum_{\kappa=1}^N b_\kappa(X,Y) v_\kappa$$

define the desired bilinear forms $b_1, \dots, b_N$. The *rank* (also called *bilinear complexity*) of $b_1, \dots, b_N$ is the smallest number of essential bilinear multiplications necessary and sufficient to compute $b_1, \dots, b_N$ from the indeterminates $X_1, \dots, X_N$ and $Y_1, \dots, Y_N$. More precisely, the bilinear complexity of $b_1, \dots, b_N$ is the smallest number $r$ of products $p_\rho = f_\rho(X) \cdot g_\rho(Y)$ with linear forms $f_\rho$ and $g_\rho$ in the $X_i$ and $Y_j$, respectively, such that $b_1, \dots, b_N$ are contained in the linear span of $p_1, \dots, p_r$. From this definition, it is obvious that the bilinear complexity of $b_1, \dots, b_N$ is independent of the choice of $v_1, \dots, v_N$; thus we may speak about the rank $R(A)$ (also called bilinear complexity) of (the multiplication in) $A$. For a modern introduction to this topic and to algebraic complexity theory in general, we recommend [9].

A fundamental lower bound for the rank of an associative algebra $A$ is the so-called Alder–Strassen bound [1]. It states that the rank of $A$ is bounded by

$$R(A) \geq 2\dim A - t, \tag{1}$$

[*]Received by the editors December 3, 2003; accepted for publication (in revised form) July 28, 2004; published electronically December 1, 2004. Parts of this work were presented at the 17th Annual IEEE Computational Complexity Conference, Montreal, 2002 [5], and at the 19th Annual Symposium on Theoretical Aspects of Computer Science, Berlin, 2003 [6].

http://www.siam.org/journals/sicomp/34-2/43827.html

[†]Institut für Theoretische Informatik, ETH Zürich, 8092 Zürich, Switzerland (mblaeser@inf.ethz.ch).

where $t$ is the number of maximal twosided ideals in $A$. This bound is tight in the sense that there are algebras for which equality holds. Such algebras are called *algebras of minimal rank*. These are the structures that allow the most efficient multiplication. While the property that (1) is tight of course completely characterizes the algebras of minimal rank, it is desirable to have an algebraic characterization, too, i.e., a characterization in terms of their algebraic structure. A lot of effort has been spent to achieve such an algebraic characterization. There has been some success for certain classes of algebras, like division algebras, commutative algebras, etc., but up to now, the general case is still unsolved. As the main contribution of the present work, we determine *all* algebras of minimal rank over *arbitrary* fields, thus solving this problem completely.

**1.1. Previous results.** One prominent algebra of minimal rank is $k^{2\times 2}$, the algebra of $2 \times 2$-matrices [20]. It has been a longstanding open problem whether $k^{3\times 3}$ is of minimal rank or not; see [9, Problem 17.1]. The idea was that if one could characterize all algebras of minimal rank in terms of their algebraic structure, then one would simply have to verify whether $k^{3\times 3}$ has this structure or not. Meanwhile, we know that $k^{3\times 3}$ is not of minimal rank [3]. Nevertheless, the characterization of the algebras of minimal rank is an interesting and important topic in algebraic complexity theory; see, e.g., [21, section 12, Problem 4] or [9, Problem 17.5].

De Groote [14] determined all division algebras $D$ of minimal rank. Over infinite fields, these are all simply generated extension fields of $k$. If $k$ is finite, then $D$ has minimal rank if, in addition, $\#k \geq 2\dim D - 2$; the latter result follows from the classification of the algorithm variety of polynomial multiplication modulo some irreducible polynomial by Winograd [22]. De Groote and Heintz [16] characterize all commutative algebras of minimal rank over infinite fields. Next Büchi and Clausen [8] describe all local algebras of minimal rank over infinite fields. Then Heintz and Morgenstern [18] determine all basic algebras over algebraically closed fields. Finally, all semisimple algebras of minimal rank over arbitrary fields and all algebras of minimal rank over algebraically closed fields have been characterized [4]: Semisimple algebras of minimal rank are isomorphic to a finite direct product of division algebras of minimal rank (as described by de Groote and Winograd) and of copies of $k^{2\times 2}$. Algebras of minimal rank over algebraically closed fields are isomorphic to a direct product of copies $k^{2\times 2}$ and a basic algebra of minimal rank (as characterized by Heintz and Morgenstern).

**1.2. New results.** As our main result, we characterize all algebras of minimal rank over arbitrary fields (Theorem 35). After more than two decades, this completely answers a major open problem in algebraic complexity theory.

An algebra $A$ over an arbitrary field $k$ is an algebra of minimal rank iff

$$A \cong C_1 \times \cdots \times C_s \times \underbrace{k^{2\times 2} \times \cdots \times k^{2\times 2}}_{u \text{ times}} \times B, \tag{2}$$

where $C_1, \ldots, C_s$ are local algebras of minimal rank with $\dim(C_\sigma / \operatorname{rad} C_\sigma) \geq 2$ (as characterized by Büchi and Clausen) and $B$ is a superbasic algebra of minimal rank. Any of the integers $s$ and $u$ may be zero, and the factor $B$ is optional. This decomposition is unique (up to permutations and isomorphms), as there are essentially three types of algebras involved and each of them has a different image modulo $\operatorname{rad} A$.

A local algebra $C_\sigma$ with $\dim(C_\sigma / \operatorname{rad} C_\sigma) \geq 2$ is of minimal rank iff $C_\sigma \cong k[X]/(p_\sigma(X)^{d_\sigma})$ for some irreducible polynomial $p_\sigma$ with $\deg p_\sigma \geq 2$, $d_\sigma \geq 1$ and the ground field $k$ fulfills $\#k \geq 2\dim C_\sigma - 2$.

An algebra $B$ is called superbasic if $B/\operatorname{rad} B = k^t$ for some $t$. A superbasic algebra $B$ is of minimal rank iff there exist $w_1, \dots, w_m \in \operatorname{rad} B$ with $w_i^2 \neq 0$ and $w_i w_j = 0$ for $i \neq j$ such that

$$\operatorname{rad} B = \mathsf{L}_B + Bw_1B + \cdots + Bw_mB = \mathsf{R}_B + Bw_1B + \cdots + Bw_mB$$

and $k$ fulfills $\#k \geq 2N(B) - 2$, where $N(B)$ denotes the largest natural number $s$ such that $(\operatorname{rad} B)^s \neq \{0\}$. Here $\mathsf{L}_B$ and $\mathsf{R}_B$ denote the left and right annihilator of $\operatorname{rad} B$ (see section 2.1 for exact definitions). The integer $m$ may be zero.

**1.3. Algorithmic aspects.** How can the characterization result be applied? Given a particular algebra, one can of course check by hand whether it fulfills the properties stated in our characterization result. An example for this is, for instance, given by Heintz and Morgenstern [18].

One can also consider the algorithmic variant, that is, given the structural constants with respect to some basis, decide algorithmically whether the corresponding algebra $A$ is an algebra of minimal rank. One possible route is the following. We only give a sketch since a full algorithm would be beyond the scope of this paper. The algorithm uses two black-boxes: computing a basis of the radical and computing a set of orthogonal primitive idempotents. The efficiency of the algorithm depends on how well these tasks can be performed for the class of algebras in consideration. Friedl and Rónyai [13] are the first to deal with this problem. For the latest result, see [12] and the references given there.

We first compute a basis of the radical. If we compute all possible products of two elements from this basis, we get a set of generators for $(\operatorname{rad} A)^2$. Any maximal subset of the basis of the radical that is linearly independent modulo $(\operatorname{rad} A)^2$ generates $(\operatorname{rad} A)$. Denote such a set by $g_1, \dots, g_m$. Any element in the radical can be expressed as a polynomial in $g_1, \dots, g_m$.

Next we compute a set of orthogonal primitive idempotents $w_1, \dots, w_s$ of $A$ with $w_1 + \cdots + w_s = 1$. From this, it is possible to compute a central decomposition $e_1, \dots, e_t$ of the identity in $A/\operatorname{rad} A$ such that each $e_\tau$ is the identity of a simple component of $A/\operatorname{rad} A$. Using $w_1, \dots, w_s$, these elements can be lifted to a decomposition $f_1, \dots, f_t$ of the identity in $A$.

We first check whether $e_\tau(A/\operatorname{rad} A)e_\tau$ has dimension one (i.e., is isomorphic to $k$) or dimension at least two.

In the second case, we first check whether $e_\tau(A/\operatorname{rad} A)e_\tau$ is simply generated, i.e., it has only one primitive idempotent. In this case $e_\tau(A/\operatorname{rad} A)e_\tau$ is a division algebra. Then we check whether $f_\tau A f_\tau$ is simply generated. Since everything is finite dimensional, we just can choose a random element and check whether it generates the radical, i.e., whether it generates a vector space whose dimension equals $\dim(f_\tau A f_\tau)$. (Here a random element means that we choose the coefficients in the linear combination of basis vectors uniformly at random from some small subset of $k$. If $f_\tau A f_\tau$ is simply generated, then this is a generator with high probability.) If $f_\tau A f_\tau$ is not simply generated, we just check whether it is isomorphic to $k^{2\times 2}$.

We are left with the first case. Let $f$ denote the sum of the $f_\tau$ corresponding to all $e_\tau$ in the first case. Then $fAf = B$. Since we know a set of generators for $\operatorname{rad} A$, we can compute a set of generators for $B \cap \operatorname{rad} A$ by multiplying by $f$ from the left and the right. We now try to find a set of generators such that one of them annihilates all the others. Once we achieve this, we can proceed inductively. We replace our generators $g_1, \dots, g_q$ by $g_1, g_2 - \alpha_2 g_1, \dots, g_q - \alpha_q g_1$ and check whether there are choices of $\alpha$ such that $g_1 \cdot (g_i - \alpha_i g_1)$ is contained in the ideal generated by $g_2, \dots, g_q$. Let $g_i' = g_i - \alpha_i g_2$. Then we try to find scalars $\beta_2, \dots, \beta_q$ such that $g_1 - \beta_2 g_2' - \dots - \beta_q g_q'$ now annihilates all the $g_i'$. (Everything is linear algebra here.) Finally we take all the generators of $\operatorname{rad} B$ whose square is nonzero, compute the left and right annihilators of $\operatorname{rad} B$, and check whether $\operatorname{rad} A$ can be written as in the characterization result.

**1.4. Organization of the paper.** In section 2, we first recall some basic facts about the structure of associative algebras. Then we describe the model of bilinear complexity. Finally, we briefly mention some important lower bounds, most of them due to Alder and Strassen, to which we will refer frequently in our proofs.

In section 3, we characterize superbasic algebras of minimal rank. We follow ideas used by Heintz and Morgenstern over algebraically closed fields and give a smoothened and simplified proof.

The proof of the preliminary characterization results over algebraically closed [4] and perfect [5] fields made use of the existence of a subalgebra $B$ of $A$ with $B \oplus \operatorname{rad} A = A$ and $B \cong A/\operatorname{rad} A$. Over perfect (and henceforth over algebraically closed) fields, such an algebra always exists by the so-called Wedderburn–Malcev theorem. Over a nonperfect field, there are examples for which such an algebra does not exist. In section 4, we show that there always exists a subalgebra $B' \subseteq A$ that is sufficiently "similar" to the algebra $B$. Using this algebra $B'$, we finally prove our main result in section 5.

## 2. Preliminaries.

**2.1. Structure of associative algebras.** We collect some elementary properties of associative algebras. The term *algebra* always means a finite dimensional associative algebra with identity 1 over some field $k$. The terms *left module* and *right module* always mean a finitely generated left module and right module, respectively, over some algebra $A$. By the embedding $\alpha \mapsto \alpha \cdot 1$, $k$ becomes a subalgebra of $A$. Hence, every $A$-left module and $A$-right module is also a finite dimensional $k$-vector space. If we speak of a basis of an algebra or a module, we always mean a basis of the underlying vector space. Further material as well as proofs of the mentioned properties can be found in [19, 10, 11].

A left ideal $I$ (and in the same way, a right ideal or twosided ideal) is called *nilpotent* if $I^n = \{0\}$ for some positive integer $n$.

FACT 1. *For all finite dimensional algebras $A$ the following hold:*

1. *The sum of all nilpotent left ideals of $A$ is a nilpotent twosided ideal, which contains every nilpotent right ideal of $A$. This twosided ideal is called the* radical *of $A$ and is denoted by* $\operatorname{rad} A$.
2. *The quotient algebra $A/\operatorname{rad} A$ contains no nilpotent ideals other than the zero ideal.*
3. *The radical of $A$ is contained in every maximal twosided ideal of $A$.*
4. *The algebras $A$ and $A/\operatorname{rad} A$ have the same number of maximal twosided ideals.*

We call an algebra $A$ *semisimple* if $\operatorname{rad} A = \{0\}$. By the above fact, $A/\operatorname{rad} A$ is semisimple. An algebra $A$ is called *simple* if there are no twosided ideals in $A$ except the zero ideal and $A$ itself.

We now describe some of the most important ways to construct new algebras from given ones: If $A$ and $B$ are $k$-algebras, then the direct product $A \times B$ with componentwise addition and multiplication is again a $k$-algebra. The set of all $n \times n$-matrices with entries from $A$ forms a $k$-algebra (with the usual definition of addition and multiplication of matrices). This algebra is denoted by $A^{n\times n}$.

We denote the set of all units of an algebra $A$, that is, the set of all invertible elements, by $A^\times$. An algebra $D$ is called a *division algebra* if $D^\times = D \setminus \{0\}$. An algebra $A$ is called *local* if $A/\operatorname{rad} A$ is a division algebra, and $A$ is called *basic* if $A/\operatorname{rad} A$ is a direct product of division algebras. Since we do not know a better name, we call $A$ *superbasic* if $A/\operatorname{rad} A \cong k^t$ for some $t$.

For an algebra $A$, $\mathsf{L}_A$ and $\mathsf{R}_A$ denote the *left and right annihilator* of $\operatorname{rad} A$; that is,

$$\begin{aligned} \mathsf{L}_A &= \{x \in \operatorname{rad} A \mid x(\operatorname{rad} A) = \{0\}\} \quad \text{and} \\ \mathsf{R}_A &= \{x \in \operatorname{rad} A \mid (\operatorname{rad} A)x = \{0\}\}. \end{aligned}$$

If $x \in A$, we denote by $AxA$ the ideal generated by $x$. If $A$ is commutative, we will also write $(x)$ for short. Furthermore, $k[x]$ denotes the smallest subalgebra of $A$ that contains $x$. If $x_1, \dots, x_m \in A$ mutually commute, then $k[x_1, \dots, x_m]$ denotes the smallest subalgebra of $A$ that contains $x_1, \dots, x_m$. For elements $v_1, \dots, v_n$ of some vector space, $\operatorname{lin}\{v_1, \dots, v_n\}$ denotes their linear span. Occasionally, we will denote this span also by $kv_1 + \cdots + kv_n$.

The following fundamental theorem describes the structure of semisimple algebras.

THEOREM 2 (Wedderburn). *Every finite dimensional semisimple algebra is isomorphic to a finite direct product of simple algebras. Every finite dimensional simple $k$-algebra $A$ is isomorphic to an algebra $D^{n\times n}$ for an integer $n \geq 1$ and a $k$-division algebra $D$. The integer $n$ and the algebra $D$ are uniquely determined by $A$ (the latter up to isomorphism).*

Wedderburn's theorem holds in a similar manner for modules over simple algebras. If $A$ is an algebra, let $A^{n\times m}$ denote the vector space of all $n \times m$-matrices with entries from $A$.

THEOREM 3 (Wedderburn). *Let $A$ be a simple algebra with $A \cong D^{n\times n}$ for some division algebra $D$. For every $A$-left module $M \neq \{0\}$ there is a (unique) integer $m \geq 1$ such that $M$ is isomorphic to $D^{n\times m}$.*

If $C$ and $D$ are algebras and $M$ is a $C$-left module that is also a $D$-right module, then the module $M$ is called a *$(C, D)$-bimodule* if, in addition, $(am)b = a(mb)$ for all $a \in C$, $m \in M$, and $b \in D$. If $C = D$, $M$ is also called a $C$-bimodule for short.

We will also need the following consequence of Nakayama's lemma. It is a useful criterion for checking whether an element in an algebra is invertible or not.

LEMMA 4 (Nakayama). *An element $x$ of an associative algebra $A$ is invertible iff its image under the canonical projection in $A/\operatorname{rad} A$ is invertible.*

This version of Nakayama's lemma can be easily derived from the more general version [11, Lemma, 3.1.4].

$A/\operatorname{rad} A$ is a semisimple algebra. It is an interesting question whether $A/\operatorname{rad} A$ is also a subalgebra of $A$. This question is answered by the Wedderburn–Malcev theorem. For its proof, see [11, Theorem 6.2.1].

THEOREM 5 (Wedderburn–Malcev). *If $A/\operatorname{rad} A$ is a separable algebra, then there is a subalgebra $S \subseteq A$ such that $S \cong A/\operatorname{rad} A$ and $S \oplus \operatorname{rad} A = A$.*

It is not necessary to define precisely what a separable algebra is. We only need the following two sufficient conditions: Every algebra over a perfect field is separable. If an algebra is of the form $k^{n_1\times n_1} \times \cdots \times k^{n_t\times n_t}$, then it is separable.

**2.2. Model of computation.** We use a coordinate-free definition of rank, which is appropriate when dealing with algebras of minimal rank; see [9, Chapter 14]. For a vector space $V$, $V^*$ denotes the dual space of $V$, that is, the vector space of all linear forms on $V$.

DEFINITION 6. *Let $k$ be a field; $U$, $V$, and $W$ be finite dimensional vector spaces over $k$; and $\phi : U \times V \to W$ be a bilinear map.*

1. *A sequence $\beta = (f_1, g_1, w_1, \dots, f_r, g_r, w_r)$ such that $f_\rho \in U^*$, $g_\rho \in V^*$, and $w_\rho \in W$ is called a* bilinear computation *of length $r$ for $\phi$ if*

$$\phi(u, v) = \sum_{\rho=1}^{r} f_\rho(u) g_\rho(v) w_\rho \quad \textit{for all } u \in U, v \in V.$$

2. *The length of a shortest bilinear computation for $\phi$ is called the* bilinear complexity *or the* rank *of $\phi$ and is denoted by $R(\phi)$.*
3. *If $A$ is a finite dimensional associative $k$-algebra, then the rank of $A$ is defined as the rank of the multiplication map of $A$, which is a bilinear map $A \times A \to A$. The rank of $A$ is denoted by $R(A)$.*

Let $\beta = (f_1, g_1, w_1, \dots, f_r, g_r, w_r)$ be a bilinear computation for an algebra $A$. Let $a, b, c \in A^\times$. We have

$$xy = a^{-1}(axb^{-1})(byc^{-1})c = \sum_{\rho=1}^{r} f_\rho(axb^{-1}) g_\rho(byc^{-1}) a^{-1} w_\rho c$$

for all $x, y \in A$. Therefore $\tilde{\beta} = (\tilde{f}_1, \tilde{g}_1, \tilde{w}_1, \dots, \tilde{f}_r, \tilde{g}_r, \tilde{w}_r)$ is a bilinear computation for $A$, where $\tilde{f}_\rho$, $\tilde{g}_\rho$, and $\tilde{w}_\rho$ are defined by $\tilde{f}_\rho(x) = f_\rho(axb^{-1})$ for all $x \in A$, $\tilde{g}_\rho(y) = g_\rho(byc^{-1})$ for all $y \in A$, and $\tilde{w}_\rho = a^{-1} w_\rho c$. This defines an equivalence relation on the set of all computations of length $r$ for $A$. This process of replacing $\beta$ by $\tilde{\beta}$ is called *sandwiching.*

We will mainly use sandwiching in the following situation: Assume that the linear forms $f_1, \dots, f_N$ of $\beta$ form a basis of $A^*$. Let $x_1, \dots, x_N$ be the dual basis of $f_1, \dots, f_N$. Now $a^{-1}x_1 b, \dots, a^{-1}x_N b$ is the dual basis of the linear forms $\tilde{f}_1, \dots, \tilde{f}_N$ of $\tilde{\beta}$. Sandwiching allows us to replace $x_1, \dots, x_N$ by $a^{-1}x_1 b, \dots, a^{-1}x_N b$.

**2.3. Lower bound techniques.** Next, we gather some of the lower bound techniques utilized by Alder and Strassen. Beside the original paper [1], [15, Chapter IV.2] and [9, Chapter 17] are excellent treatments of the results of Alder and Strassen. We have taken the term *separate* and the extension lemma from those sources, but everything is also contained in the work of Alder and Strassen [1].

DEFINITION 7. *Let $U$, $V$, and $W$ be vector spaces over some field $k$, and let $\beta = (f_1, g_1, w_1, \dots, f_r, g_r, w_r)$ be a computation for a bilinear map $\phi : U \times V \to W$. Let $U_1 \subseteq U$, $V_1 \subseteq V$, and $W_1 \subseteq W$ be subspaces. The computation $\beta$* separates $(U_1, V_1, W_1)$ *if there are disjoint sets of indices $I, J \subseteq \{\rho \mid w_\rho \notin W_1\}$ such that*

$$U_1 \cap \bigcap_{i \in I} \ker f_i = \{0\} \quad \textit{and} \quad V_1 \cap \bigcap_{j \in J} \ker g_j = \{0\}.$$

The latter condition is equivalent to the condition that $(f_i|_{U_1})_{i \in I}$ and $(g_j|_{V_1})_{j \in J}$ generate the dual spaces $U_1^*$ and $V_1^*$, respectively.

If $\phi : U \times V \to W$ is a bilinear map and $U_1 \subseteq U$ and $V_1 \subseteq V$ are subspaces, then $(u + U_1, v + V_1) \mapsto \phi(u, v) + \tilde{W}$ defines a bilinear map $U/U_1 \times V/V_1 \to W/\tilde{W}$, where

$\tilde{W} := \mathrm{lin}\{\phi(U_1, V) + \phi(U, V_1)\}$. We denote this map by $\phi/(U_1 \times V_1)$. It is called the *quotient* of $\phi$ by $U_1$ and $V_1$. The following lemma (see [9] for a proof) provides a lower bound for the rank of a bilinear map in terms of the rank of a quotient of it.

LEMMA 8. *Let $U$, $V$, and $W$ be vector spaces, and let $\beta = (f_1, g_1, w_1, \dots, f_r, g_r, w_r)$ be a bilinear computation for some bilinear map $\phi : U \times V \to W$. Let $U_1 \subseteq U$, $V_1 \subseteq V$, and $W_1 \subseteq W$ be subspaces such that $\beta$ separates $(U_1, V_1, W_1)$. Let $\pi$ be an endomorphism of $W$ such that $W_1 \subseteq \ker \pi$. Then*

$$r \geq R\,(\pi \circ \phi/U_1 \times V_1) + \dim U_1 + \dim V_1 + \#\{\rho \mid w_\rho \in W_1\}.$$

COROLLARY 9. *Let $A$ be an algebra and $I \subseteq A$ a twosided ideal. Let $\beta$ be a bilinear computation for $A$ of length $r$ that separates $(I, I, \{0\})$. Then*

$$r \geq R\,(A/I) + 2 \dim I.$$

Together with Corollary 9, the next lemma yields the first of the two important bounds established by Alder and Strassen, namely, for any algebra $A$

$$R(A) \geq R\,(A/\mathrm{rad}\, A) + 2 \dim A. \tag{3}$$

LEMMA 10. *Let $\beta$ be a bilinear computation for an associative algebra $A$. Then $\beta$ separates* $(\mathrm{rad}\, A, \mathrm{rad}\, A, \{0\})$.

Having dealt with the radical, Alder and Strassen turn to semisimple algebras. Their second important result is subsumed in the following lemma.

LEMMA 11. *If $A = B \times B'$, with $B$ being a simple $k$-algebra and $B'$ being an arbitrary $k$-algebra, then*

$$R(A) \geq 2 \dim B - 1 + R(B').$$

To achieve good lower bounds by means of Lemma 8, one has to find an optimal bilinear computation that separates a "large" triple. An important tool for solving this task is the following "extension lemma," which is essentially due to Alder and Strassen.

LEMMA 12. *Let $U$, $V$, $W$ be vector spaces over a field $k$, and let $\beta$ be a computation for a bilinear map $\phi : U \times V \to W$. Let $U_1 \subseteq U_2 \subseteq U$, $V_1 \subseteq V$, and $W_1 \subseteq W$ be subspaces such that $\beta$ separates the triple $(U_1, V_1, W_1)$. Then $\beta$ separates also $(U_2, V_1, W_1)$, or there is some $u \in U_2 \setminus U_1$ such that*

$$\phi(u, V) \subseteq \mathrm{lin}\{\phi(u, V_1)\} + W_1.$$

**3. Superbasic algebras of minimal rank.** We characterize superbasic algebras of minimal rank, thus generalizing the results of Heintz and Morgenstern [18]. (Note that, over algebraically closed fields, the notions of basic and superbasic coincide, since there are no proper division algebras over algebraically closed fields.) Our main result is Theorem 22. It turns out that, over infinite fields, superbasic algebras of minimal rank have the same structure as (super)basic algebras over algebraically closed fields. If the underlying field $k$ is finite, then a superbasic algebra $A$ of minimal rank also fulfils $\#k \geq 2N(A) - 2$, where $N(A)$ denotes the largest natural number $s$ such that $(\mathrm{rad}\, A)^s \neq \{0\}$. (Since $A$ is finite dimensional, this is well defined.)

Inspired by Heintz and Morgenstern, we introduce the class $\mathsf{M}_k$ of superbasic algebras as an intermediate concept. This class plays an important role in our proof.

DEFINITION 13. *Let $A$ be a superbasic algebra over some field $k$ with* $\dim A = n$ *and* $\dim A/\mathrm{rad}\, A = t$. *The algebra $A$ belongs to the class $\mathsf{M}_k$ if there are bases $x_1, \dots, x_n$ and $y_1, \dots, y_n$ of $A$ such that*

1. *$x_\mu y_\nu \in kx_\mu + ky_\nu$ for $t+1 \le \mu, \nu \le n$,*
2. *$x_\tau = y_\tau$ for $1 \le \tau \le t$,*
3. *$x_\tau^2 = x_\tau$ and $x_\sigma x_\tau = 0$ for $1 \le \sigma, \tau \le t$, and $\tau \ne \sigma$,*
4. *$x_1 + \cdots + x_t = 1$, and*
5. *$x_\sigma y_\nu \in ky_\nu$ and $x_\mu y_\tau \in kx_\mu$ for $1 \le \sigma, \tau \le t$ and $t+1 \le \mu, \nu \le n$.*

*Such two bases are called an* $\mathsf{M}$*-pair of bases for $A$.*

Superbasic algebras of minimal rank are in $\mathsf{M}_k$.

LEMMA 14. *If a superbasic algebra $A$ has minimal rank, then $A$ is in $\mathsf{M}_k$.*

*Proof.* Assume that $\dim A = n$ and $\dim A/\operatorname{rad} A = t$. Moreover, let $\beta = (f_1, g_1, w_1, \dots, f_{2n-t}, g_{2n-t}, w_{2n-t})$ be an optimal computation for $A$. By permuting the products, we can achieve that $f_1, \dots, f_n$ form a basis of $A^*$. Furthermore, we may assume that for $S := \bigcap_{\nu=1}^{n-t} \ker f_\nu$, $S + \operatorname{rad} A = A$. $S$ contains an invertible element $e$ of $A$ by Nakayama's lemma (Lemma 4).

Thus $g_{n-t+1}, \dots, g_{2n-t}$ form a basis of $A^*$, because otherwise there would be some nonzero $a \in \bigcap_{\nu=1}^{n} \ker g_{n-t+\nu}$. But then

$$e \cdot a = \sum_{\rho=1}^{2n-t} f_\rho(e) g_\rho(a) w_\rho = 0,$$

which is a contradiction. By permuting the products $n-t+1, \dots, 2n-t$, we can achieve that for $T := \bigcap_{\nu=n+1}^{2n-t} \ker g_\nu$, $T + \operatorname{rad} A = A$. This permutation affects neither $f_1, \dots, f_{n-t}$ nor $S$. In particular, we still have $e \in \bigcap_{\nu=1}^{n-t} \ker f_\nu$. Again by Nakayama's lemma (Lemma 4), $T$ contains an invertible element $e'$ of $A$. With $e'$ instead of $e$, we can conclude that the new $f_1, \dots, f_n$ still form a basis of $A^*$, in the same way we have shown that $g_{n-t+1}, \dots, g_{2n-t}$ is a basis.

By sandwiching with $e^{-1}$ from the left and $(e')^{-1}$ from the right (see section 2.2), we may assume that $e = e' = 1$. Let $x_{t+1}, \dots, x_n, x_1, \dots, x_t$ denote the dual basis of $f_1, \dots, f_n$. (The odd numbering will become clear in a moment.) Furthermore, let $y_1, \dots, y_n$ be the dual basis of $g_{n-t+1}, \dots, g_{2n-t}$. By construction, $x_1, \dots, x_t$ span $S$ and $y_1, \dots, y_t$ span $T$. The properties of the $f_1, \dots, f_{2n-t}$, $g_1, \dots, g_{2n-1}$, and the dual bases $x_1, \dots, x_n$ as well as $y_1, \dots, y_n$, can be depicted as follows:

$$\begin{array}{cccccccccc}
f_1 & \cdots & f_{n-t} & f_{n-t+1} & \cdots & f_n & f_{n+1} & \cdots & f_{2n-t} & \\
x_1 & \cdots & x_{n-t} & x_1 & \cdots & x_t & & & & \text{(dual basis)} \\
 & & & & & & & & & \\
g_1 & \cdots & g_{n-t} & g_{n-t+1} & \cdots & g_n & g_{n+1} & \cdots & g_{2n-t} & \\
 & & & y_1 & \cdots & y_t & y_{t+1} & \cdots & y_n & \text{(dual basis).}
\end{array}$$

By appropriate scaling, we can achieve that $1 = \delta_1 x_1 + \cdots + \delta_t x_t$ with $\delta_\tau \in \{0, 1\}$. Since

$$1 \cdot y = \sum_{\tau=1}^{t} \delta_\tau g_{n-t+\tau}(y) w_{n-t+\tau} + \sum_{\nu=1}^{n-t} f_{n+\nu}(1) g_{n+\nu}(y) w_{n+\nu}$$

for any $y \in A$,

$$A = \operatorname{lin}\{\delta_1 w_{n-t+1}, \dots, \delta_t w_n, w_{n+1}, \dots, w_{2n-t}\}.$$

Thus $\delta_\tau = 1$ for all $1 \le \tau \le t$ by a dimension argument. The same reasoning shows $1 = y_1 + \cdots + y_t$. This establishes the fourth condition of the $\mathsf{M}$-pair for $x_1, \dots, x_n$ and $y_1, \dots, y_n$.

By the construction of $S$ and $T$ and the definition of dual basis,

$$x_\nu = x_\nu \cdot 1 = \sum_{\rho=1}^{2n-t} f_\rho(x_\nu) g_\rho(1) w_\rho = \underbrace{g_{\nu-t}(1)}_{\neq 0} w_{\nu-t}$$

and

$$y_\nu = 1 \cdot y_\nu = \sum_{\rho=1}^{2n-t} f_\rho(1) g_\rho(y_\nu) w_\rho = \underbrace{f_{n-t+\nu}(1)}_{\neq 0} w_{n-t+\nu}$$

for all $t+1 \leq \nu \leq n$. In other words, $x_\nu$ and $w_{\nu-t}$ as well as $y_\nu$ and $w_{n-t+\nu}$ are multiples of each other, respectively. This yields

$$\begin{aligned} x_\mu y_\nu &= \sum_{\rho=1}^{2n-t} f_\rho(x_\mu) g_\rho(y_\nu) w_\rho \\ &= g_{\mu-t}(y_\nu) w_{\mu-t} + f_{n-t+\nu}(x_\mu) w_{n-t+\nu} \\ &\in kx_\mu + ky_\nu \end{aligned}$$

for $t+1 \leq \mu, \nu \leq n$, which shows that our bases fulfil the first condition.

The second condition follows from the fact that

$$x_\tau = x_\tau \cdot 1 = w_{n-t+\tau} \quad \text{and} \quad y_\tau = 1 \cdot y_\tau = w_{n-t+\tau}$$

for $1 \leq \tau \leq t$. (Note that since $1 = y_1 + \cdots + y_t$, $g_{n-t+\tau}(1) = 1$. $f_{n-t+\tau}(1) = 1$ follows similarly.)

This also establishes the third condition, as for $\tau \neq \sigma$,

$$x_\tau^2 = x_\tau y_\tau = w_{n-t+\tau} = x_\tau \quad \text{and} \quad x_\sigma x_\tau = x_\sigma y_\tau = 0.$$

It remains to show that the two bases meet the fifth condition: This is again shown by exploiting the property of dual bases. We have

$$x_\sigma y_\nu = f_{n-t+\nu}(x_\sigma) w_{n-t+\nu} \in ky_\nu$$

for $1 \leq \sigma \leq t$ and $t+1 \leq \nu \leq n$. The fact $x_\mu y_\tau \in kx_\mu$ follows in the same manner. □

The bases of an $\mathsf{M}$-pair can be further normalized.

Definition 15. *A superbasic algebra in $\mathsf{M}_k$ has a normalized $\mathsf{M}$-pair of bases $x_1, \ldots, x_n$ and $y_1, \ldots, y_n$ if, in addition to the conditions in Definition 13, the bases $x_1, \ldots, x_n$ and $y_1, \ldots, y_n$ fulfil*

1. $x_{t+1}, \ldots, x_n \in \operatorname{rad} A$ *and* $y_{t+1}, \ldots, y_n \in \operatorname{rad} A$,
2. $x_{n-d+1}, \ldots, x_n$ *and* $y_{n-d+1}, \ldots, y_n$ *are both bases of* $\mathsf{L}_A \cap \mathsf{R}_A$, *where* $d = \dim \mathsf{L}_A \cap \mathsf{R}_A$, *and*
3. $x_{n-\ell+1}, \ldots, x_n$ *is a basis of* $\mathsf{L}_A$ *and* $y_{n-r+1}, \ldots, y_n$ *is a basis of* $\mathsf{R}_A$, *where* $\ell = \dim \mathsf{L}_A$ *and* $r = \dim \mathsf{R}_A$.

Lemma 16. *Every superbasic algebra $A \in \mathsf{M}_k$ has a normalized $\mathsf{M}$-pair of bases.*

*Proof.* Since $A \in \mathsf{M}_k$, it has an $\mathsf{M}$-pair of bases $x_1, \ldots, x_n$ and $y_1, \ldots, y_n$. By the third condition, $x_1, \ldots, x_t$ are a system of mutually orthogonal idempotents of $A$, none of them being zero; thus their images $\bar{x}_1, \ldots, \bar{x}_t$ under the canonical projection

$A \to A/\operatorname{rad} A$ are linearly independent. Comparing dimensions shows that they even form a basis. Thus for any $t+1 \le \mu \le n$ there are scalars $\xi_{\mu,\tau}$ such that

$$x'_\mu := x_\mu - \sum_{\tau=1}^{t} \xi_{\mu,\tau} x_\tau \in \operatorname{rad} A.$$

We define scalars $\eta_{\nu,\tau}$ and elements $y'_\nu$ in the same manner. We claim that the bases $x_1, \dots, x_t, x'_{t+1}, \dots, x'_n$ and $y_1, \dots, y_t, y'_{t+1}, \dots, y'_n$ are an $\mathsf{M}$-pair. By construction, we only have to verify the first and fifth condition in Definition 13. For the first, note that

$$\begin{aligned} x'_\mu y'_\nu &= \left(x_\mu - \sum_{\tau=1}^{t} \xi_{\mu,\tau} x_\tau\right)\left(y_\nu - \sum_{\tau=1}^{t} \eta_{\nu,\tau} y_\tau\right) \\ &= \xi x_\mu + \eta y_\nu + \sum_{\tau=1}^{t} \alpha_\tau x_\tau \\ &= \xi x'_\mu + \eta y'_\nu + \sum_{\tau=1}^{t} \alpha'_\tau x_\tau \qquad (4) \end{aligned}$$

for suitable scalars $\xi, \eta, \alpha_1, \dots, \alpha_t, \alpha'_1, \dots, \alpha'_t$. Since $x'_\mu y'_\nu$, $x'_\mu$, and $x'_\nu$ are contained in $\operatorname{rad} A$ but $\operatorname{lin}\{x_1, \dots, x_t\} \cap \operatorname{rad} A = \{0\}$, $\alpha'_1 = \dots = \alpha'_t = 0$ in (4).

In a similar manner, one can verify that $x_\tau y'_\nu \in k y'_\nu$ and $x'_\nu y_\tau \in k x'_\nu$ for $1 \le \tau \le t$ and $t+1 \le \nu \le n$: We have

$$x_\tau y'_\nu = x_\tau \left(y_\nu - \sum_{\tau=1}^{t} \eta_{\nu,\tau} y_\tau\right) = \eta y'_\nu + \sum_{\tau=1}^{t} \alpha'_\tau x_\tau$$

for suitable $\eta, \alpha'_1, \dots, \alpha'_t$. Again we conclude that $\alpha'_1 = \dots = \alpha'_t = 0$. The statement $x'_\nu y_\tau \in k x'_\nu$ follows similarly. Thus $x_1, \dots, x_t, x'_{t+1}, \dots, x'_n$ and $y_1, \dots, y_t, y'_{t+1}, \dots, y'_n$ are an $\mathsf{M}$-pair. For convenience, we call these bases again $x_1, \dots, x_n$ and $y_1, \dots, y_n$, respectively.

Let $a_1, \dots, a_d$ and $b_1, \dots, b_{\ell-d}, a_1, \dots, a_d$ and $c_1, \dots, c_{r-d}, a_1, \dots, a_d$ be bases of $\mathsf{L}_A \cap \mathsf{R}_A$ and $\mathsf{L}_A$ and $\mathsf{R}_A$, respectively. We can achieve that each $a_\delta$, $b_\lambda$, and $c_\rho$ is contained in exactly one of the subspaces $x_\sigma(\mathsf{L}_A \cap \mathsf{R}_A)x_\tau$, $\mathsf{L}_A x_\tau$, and $x_\sigma \mathsf{R}_A$, respectively. We show this for $a_1, \dots, a_d$, and the other two cases follow along the same lines: We take any basis $a_1, \dots, a_d$. Since $1 = x_1 + \dots + x_t$, the set of all $x_\sigma a_\delta x_\tau$, $1 \le \sigma, \tau \le t$, $1 \le \delta \le d$, generates the same space as $a_1, \dots, a_d$. Any maximally linearly independent set of the $x_\sigma a_\delta x_\tau$ is a basis such that each element is contained in one of the subspaces $x_\sigma(\mathsf{L}_A \cap \mathsf{R}_A)x_\tau$. To see that each element is contained in exactly one such subspace, note that their pairwise intersections are the nullspace: Let $V$ be any subspace. Assume that there is an $a$ in $x_\sigma V x_\tau \cap x_{\sigma'} V x_{\tau'}$ with $\sigma \ne \sigma'$, say. (The case $\tau \ne \tau'$ is symmetric.) That means we can write $a = x_\sigma v = x_{\sigma'} v'$. But then

$$a = x_\sigma v = x_\sigma^2 v = x_\sigma x_{\sigma'} v' = 0,$$

which proves the claim.

By a suitable renumbering of $x_{t+1}, \dots, x_n$ and $y_{t+1}, \dots, y_n$, we can achieve that

$$\begin{aligned} &x_1, \dots, x_{n-\ell}, b_1, \dots, b_{\ell-d}, a_1, \dots, a_d \quad \text{and} \\ &y_1, \dots, y_{n-r}, c_1, \dots, c_{r-d}, a_1, \dots, a_d \end{aligned}$$

are bases of $A$. (We have only to renumber the $x_{t+1}, \dots, x_n$, since $x_1, \dots, x_t \notin \operatorname{rad} A$ whereas $x_{t+1}, \dots, x_n \in \operatorname{rad} A$. We want to exchange with $b_1, \dots, b_{\ell-d}, a_1, \dots, a_d$, which are all in $\operatorname{rad} A$ by the definition of the left annihilator. Hence, $x_1, \dots, x_t$ cannot be exchanged. The same holds for $y_1, \dots, y_t$.) We assert that this is an $\mathsf{M}$-pair for $A$. To prove this, we have to check the first and fifth conditions in Definition 13. The first is clearly fulfilled, since the $x_\nu$ and $y_\nu$ with $t+1 \le \nu$ are in $\operatorname{rad} A$ (since they were before) and $\mathsf{L}_A$ and $\mathsf{R}_A$ are the left and right annihilator, respectively, of $\operatorname{rad} A$.

The fifth condition is also true by construction: Each $a_\delta$ can be written as $x_\sigma a'_\delta x_\tau$ for some $a'_\delta \in \mathsf{L}_A \cap \mathsf{R}_A$. Thus for $1 \le i \le t$, $x_i a_\delta = x_i x_\sigma a'_\delta x_\tau$ is either $a_\delta$ or 0, depending on whether $i = \sigma$ or not. The same is true for $a_\delta x_i$, $b_\lambda x_i$, and $x_i c_\rho$. This completes the proof. $\square$

The next two lemmas describe the structure of the algebras in $\mathsf{M}_k$.

LEMMA 17. *Let $A \in \mathsf{M}_k$; let $x_1, \dots, x_n$ and $y_1, \dots, y_n$ be a normalized $\mathsf{M}$-pair of bases for $A$; and let $\ell = \dim \mathsf{L}_A$ and $r = \dim \mathsf{R}_A$. Then $x_\mu y_\nu = 0$ or $k[x_\mu] = k[y_\nu]$ for all $t+1 \le \mu, \nu \le n$. Moreover, $x_\mu^2 \ne 0$ and $y_\nu^2 \ne 0$ for all $1 \le \mu \le n-\ell$ and $1 \le \nu \le n-r$.*

*Proof.* If $\mu \ge n-\ell+1$ or $\nu \ge n-r+1$, the first statement of the lemma is true by the definition of $\mathsf{L}_A$ or $\mathsf{R}_A$, respectively. Thus assume that $\mu \le n-\ell$ and $\nu \le n-r$. If $x_\mu y_\nu = 0$, we are done. Otherwise, assume that

$$x_\mu y_\nu = \xi x_\mu + \eta y_\nu. \tag{5}$$

Let $m$ be the largest integer $\ge 1$ such that $x_\mu y_\nu \in (\operatorname{rad} A)^m$. Since $x_\mu, y_\nu \in \operatorname{rad} A$, neither $x_\mu \in (\operatorname{rad} A)^m$ nor $y_\nu \in (\operatorname{rad} A)^m$. Thus $\xi \ne 0$ and $\eta \ne 0$ in (5). It follows that $(x_\mu - \eta)(y_\nu - \xi) = \xi\eta \in A^\times$, and hence

$$k[x_\mu] = k[x_\mu - \eta] = k[y_\nu - \xi] = k[y_\nu].$$

For the second statement, note that since $x_\mu \notin \mathsf{L}_A$, there is an $i$ with $t+1 \le i \le n-r$ such that $x_\mu y_i \ne 0$, particularly $k[x_\mu] = k[y_i]$. As $x_\mu, y_i \in \operatorname{rad} A$ and therefore are nilpotent, $k[x_\mu]$ and $k[y_i]$ are both isomorphic to $k[X]/(X^m)$ for the same $m$. The isomorphism is given by $x_\mu \mapsto X$ and $y_i \mapsto X$. Since $k[x_\mu] = k[y_i]$, we can write $x_\mu$ as a polynomial in $y_i$ without constant term and nonvanishing linear term, i.e.,

$$x_\mu = \alpha_1 y_i + \alpha_2 y_i^2 + \dots + \alpha_m y_i^m \qquad \text{with } \alpha_1 \ne 0. \tag{6}$$

Since both $x_\mu, y_i \in \operatorname{rad} A$,

$$x_\mu^2 = \underbrace{\alpha_1 x_\mu y_i}_{\ne 0} + \text{higher order terms} \ne 0.$$

The result $y_\nu^2 \ne 0$ is proven identically. $\square$

LEMMA 18. *Let $A \in \mathsf{M}_k$; let $x_1, \dots, x_n$ and $y_1, \dots, y_n$ be a normalized $\mathsf{M}$-pair of bases for $A$; and let $\ell = \dim \mathsf{L}_A$ and $r = \dim \mathsf{R}_A$. Then either $x_\mu x_\nu = x_\nu x_\mu = 0$ or $k[x_\mu] = k[x_\nu]$ for all $t+1 \le \mu, \nu \le n-\ell$. Moreover, $x_1, \dots, x_{n-\ell}$ mutually commute. The same holds for $y_1, \dots, y_{n-r}$.*

*Proof.* We start by proving the first statement. This also shows that $x_{t+1}, \dots, x_{n-\ell}$ mutually commute. Let $x_\mu$ and $x_\nu$ with $t+1 \le \mu, \nu \le n-\ell$ be given. By assumption, there are $t+1 \le i, j \le n-r$ such that $x_\mu y_i \ne 0$ and $x_\nu y_j \ne 0$. Thus $k[x_\mu] = k[y_i]$ and $k[x_\nu] = k[y_j]$, by Lemma 17. We distinguish two cases: $k[x_\mu] \ne k[y_j]$ or $k[x_\mu] = k[y_j]$.

In the first case, also $k[x_\nu] \neq k[y_i]$, and hence $x_\mu y_j = x_\nu y_i = 0$ by Lemma 17. Since $k[x_\mu] = k[y_i]$, we can write $x_\mu$ as a polynomial in $y_i$ as in (6). Therefore,

$$x_\nu x_\mu = x_\nu(\alpha_1 y_i + \alpha_2 y_i^2 + \cdots + \alpha_m y_i^m) = 0.$$

In the same way, $x_\mu x_\nu = 0$. In the second case, obviously $k[x_\mu] = k[x_\nu]$. We have to show that $x_\mu x_\nu \neq 0$. But $x_\mu x_\nu = 0$ would mean that $x_\mu \in \mathsf{L}_A$, since $k[x_\mu] = k[x_\nu]$, a contradiction.

To prove the second statement, it suffices to show that $x_\tau x_\nu = x_\nu x_\tau$ for $1 \leq \tau \leq t$ and $t+1 \leq \nu \leq n-\ell$, since $x_1, \dots, x_t$ mutually commute by the definition of an $\mathsf{M}$-pair. Because $x_\tau = y_\tau$, we have $x_\nu x_\tau = \xi x_\nu$. Since $x_\tau$ is idempotent, $\xi \in \{0,1\}$; that is, $x_\nu x_\tau \in \{0, x_\nu\}$. On the other hand, as $x_\nu \notin \mathsf{L}_A$, there is some $y_i$ such that $x_\nu y_i \neq 0$. By Lemma 17, $k[x_\nu] = k[y_i]$. As in (6), we can write $x_\nu$ as a polynomial in $y_i$ without constant term. A similar argument as above shows that $x_\tau y_i \in \{0, y_i\}$. Together with (6), this implies $x_\tau x_\nu \in \{0, x_\nu\}$.

From $x_\tau x_\nu, x_\nu x_\tau \in \{0, x_\nu\}$, we can conclude that $x_\tau$ and $x_\nu$ commute. Assume on the contrary that, say, $x_\tau x_\nu = x_\nu$ but $x_\nu x_\tau = 0$. Then $x_\nu^2 = x_\nu(x_\tau x_\nu) = (x_\nu x_\tau)x_\nu = 0$, contradicting the second statement of Lemma 17.

In a similar fashion, one proves the statement for $y_1, \dots, y_{n-r}$. □

COROLLARY 19. *Let $A \in \mathsf{M}_k$. Then there is a commutative subalgebra $S \subseteq A$ such that $S + \mathsf{L}_A = S + \mathsf{R}_A = A$. Moreover,* $\operatorname{rad} S = (w_1) + \cdots + (w_m)$ *with nilpotent $w_1, \dots, w_m$ such that $w_i^2 \neq 0$ for all $1 \leq i \leq m$ and $w_i w_j = 0$ for $i \neq j$.*

*Proof.* Let $x_1, \dots, x_n$ and $y_1, \dots, y_n$ be a normalized $\mathsf{M}$-pair of bases of $A$. By Lemma 18, $k[x_1, \dots, x_{n-\ell}]$ is a commutative algebra. By Lemma 17, $k[y_1, \dots, y_{n-r}] = k[x_1, \dots, x_{n-\ell}]$. Hence, we set $S := k[x_1, \dots, x_{n-\ell}]$. By the definition of a normalized $\mathsf{M}$-pair, $S + \mathsf{L}_A = S + \mathsf{R}_A = A$.

We have $\operatorname{rad} S = (x_{t+1}, \dots, x_{n-\ell})$. We define an equivalence relation $\equiv$ on the elements $x_{t+1}, \dots, x_{n-\ell}$ by

$$x_\mu \equiv x_\nu \quad \text{iff} \quad x_\mu x_\nu \neq 0.$$

By Lemma 18, $k[x_\mu] = k[x_\nu]$ in this case. Take a set $w_1, \dots, w_m$ of the representatives for the $\equiv$-classes. Then $\operatorname{rad} S = (w_1, \dots, w_m) = (w_1) + \cdots + (w_m)$. We also may assume $w_i^2 \neq 0$ because otherwise $w_i \in \mathsf{L}_A \cap \mathsf{R}_A$. □

COROLLARY 20. *If $A \in \mathsf{M}_k$, then* $\dim \mathsf{L}_A = \dim \mathsf{R}_A$.

*Proof.* By Corollary 19, $A = S + \mathsf{L}_A = S + \mathsf{R}_A$ for a commutative subalgebra $S$ of $A$. As $S$ is commutative and $\mathsf{L}_A^2 = \mathsf{R}_A^2 = \{0\}$, $S \cap \mathsf{L}_A = S \cap \mathsf{R}_A$. We have

$$\dim S + \dim \mathsf{L}_A - \dim S \cap \mathsf{L}_A = \dim A = \dim S + \dim \mathsf{R}_A - \dim S \cap \mathsf{R}_A.$$

Thus $\dim \mathsf{L}_A = \dim \mathsf{R}_A$. □

The next lemma basically states that if there is an algebra $A \in \mathsf{M}_k$ with $N(A) = d$, then we can simulate univariate polynomial multiplication mod $X^{d+1}$ efficiently.

LEMMA 21. *Assume there is an $A \in \mathsf{M}_k$ with $N(A) = d$. Then $k[X]/(X^{d+1})$ has minimal rank, where $X$ denotes some indeterminate.*

*Proof.* If $d \leq 1$, then $k[X]/(X^{d+1})$ has minimal rank by using the trivial algorithm. So we may assume $d > 1$. Let $x_1, \dots, x_n$ and $y_1, \dots, y_n$ be a normalized $\mathsf{M}$-pair of bases for $A$ as asserted by Lemma 16. There is some index, say $t+1$, such that $x_{t+1}^d \neq 0$. Furthermore $x_{t+1}^d \in \mathsf{L}_A \cap \mathsf{R}_A$ by the maximality of $d$. By Lemmas 17 and 18, there are indices $i_1 = t+1, \dots, i_{d-1}$ as well as indices $j_1, \dots, j_{d-1}$ such that

$$\begin{aligned} k[x_{t+1}] &= \operatorname{lin}\{1, x_{i_1}, \dots, x_{i_{d-1}}, x_{t+1}^d\} \\ &= \operatorname{lin}\{1, y_{j_1}, \dots, y_{j_{d-1}}, x_{t+1}^d\}. \end{aligned}$$

We claim that $R(k[x_{t+1}]) = 2d + 1$: By the definition of an $\mathsf{M}$-pair,

$$x_\mu y_\nu = \xi_{\mu,\nu} x_\mu + \eta_{\mu,\nu} y_\nu \tag{7}$$

for suitable scalars $\xi_{\mu,\nu}$ and $\eta_{\mu,\nu}$.

Define a computation

$$(f_1, g_1, w_1, \dots, f_{2d+1}, g_{2d+1}, w_{2d+1}) \quad \text{for } k[x_{t+1}]$$

as follows: As the elements $w_1, \dots, w_{2d+1}$, we choose $1, x_{i_1}, \dots, x_{i_{d-1}}, x_{t+1}^d$, $y_{j_1}, \dots, y_{j_{d-1}}, x_{t+1}^d$. Let $f_1, \dots, f_{d+1}$ be the dual basis of $1, x_{i_1}, \dots, x_{i_{d-1}}, x_{t+1}^d$ and let $g_1, g_{d+2}, \dots, g_{2d+1}$ be the dual basis of $1, y_{j_1}, \dots, y_{j_{d-1}}, x_{t+1}^d$. Next the $f_\delta$ with $d+2 \le \delta \le 2d$ are defined by $f_\delta(x_{i_\tau}) = \eta_{i_\tau, j_\delta}$ for $1 \le \tau \le d-1$ and $f_\delta(1) = f_\delta(x_{t+1}^d) = 0$. The $g_\delta$ with $2 \le \delta \le d$ are defined accordingly. Finally, $f_{2d+1}$ is defined by $f_{2d+1}(x_{t+1}^d) = 0$, $f_{2d+1}(1) = 1$, and $f_{2d+1}(x_{j_\tau}) = 0$ for $1 \le \tau \le d-1$. The linear form $g_{d+1}$ is defined in the same manner.

Exploiting (7), it is easy to see that this defines a computation for $k[x_{t+1}]$. Noting that $k[x_{t+1}] \cong k[X]/(X^{d+1})$ completes the proof. □

THEOREM 22. *Let $A$ be a superbasic algebra. Then the following statements are equivalent:*

1. *$A$ has minimal rank.*
2. *$A \in \mathsf{M}_k$.*
3. *There exist $w_1, \dots, w_m \in \operatorname{rad} A$ with $w_i^2 \neq 0$ for all $i$ and $w_i w_j = 0$ for $i \neq j$ such that*

$$\begin{aligned} \operatorname{rad} A &= \mathsf{L}_A + A w_1 A + \dots + A w_m A \\ &= \mathsf{R}_A + A w_1 A + \dots + A w_m A \end{aligned}$$

*and $\#k \ge 2N(A) - 2$.*

*Proof.* 1. ⇒ 2.: This follows at once from Lemma 14.

2. ⇒ 3.: Corollary 19 implies the first part of 3. For the second part, let $d = N(A)$. By Lemma 21, the algebra $k[X]/(X^{d+1})$ has minimal rank. From the classification of the algorithm variety of $k[X]/(X^{d+1})$ given by Averbuch, Galil, and Winograd [2], it follows that this can be the case only if $\#k \ge 2d - 2$.

3. ⇒ 1.: Let $\bar{A} = A/\operatorname{rad} A = k^t$ and let $\bar{e}_1, \dots, \bar{e}_t$ denote a decomposition of the identity of $A/\operatorname{rad} A$. By [11, Corollary 3.3.9], we can lift this decomposition to a decomposition $e_1, \dots, e_t$ of the identity of $A$. In particular, $\operatorname{lin}\{e_1, \dots, e_t\}$ forms a subalgebra $S$ of $A$. Consider $\mathsf{L}_A$ and $\mathsf{R}_A$ as $S$-right module and $S$-left module, respectively. Since $S$ is semisimple, $\mathsf{L}_A$ and $\mathsf{R}_A$ are semisimple modules, too. Hence there are bases $b_1, \dots, b_\ell$ and $c_1, \dots, c_r$ of $\mathsf{L}_A$ and $\mathsf{R}_A$, respectively, such that

$$b_\lambda A = k b_\lambda \quad \text{for all } 1 \le \lambda \le \ell \qquad \text{and} \qquad A c_\rho = k c_\rho \quad \text{for all } 1 \le \rho \le r. \tag{8}$$

By Corollary 20, $\ell = r$. Choose $q_i \ge 2$ such that $w_i^{q_i} \neq 0$ and $w_i^{q_i+1} = 0$. Since each $w_\mu^{q_\mu} \in \mathsf{L}_A \cap \mathsf{R}_A$,

$$\begin{aligned} &e_1, \dots, e_t, w_1, \dots, w_1^{q_1-1}, \dots, w_m, \dots, w_m^{q_m-1}, b_1, \dots, b_\ell \qquad \text{and} \\ &e_1, \dots, e_t, w_1, \dots, w_1^{q_1-1}, \dots, w_m, \dots, w_m^{q_m-1}, c_1, \dots, c_\ell \end{aligned}$$

are bases of $A$. Call these bases $x_1, \dots, x_n$ and $y_1, \dots, y_n$.

Given two generic elements $\xi_1 x_1 + \cdots + \xi_n x_n$ and $\eta_1 y_1 + \cdots + \eta_n y_n$, we now have to show how to compute the coefficients (which are bilinear forms in the $\xi_i$ and $\eta_j$) of their product with $2n - t$ bilinear multiplications.

We first compute $\xi_\tau \eta_\tau$ for $1 \le \tau \le t$ with $t$ bilinear multiplication.

Since $e_1, \dots, e_t$ annihilate each other, for each $\lambda$ there is exactly one $\tau$ such that $b_\lambda e_\tau \ne 0$ by (8). In the same way, for each $\lambda$ there is exactly one $\tau$ such that $e_\tau c_\lambda \ne 0$. Thus we compute the coefficients of $b_1, \dots, b_\ell$ and $c_1, \dots, c_\ell$ with $2\ell$ bilinear products.

For each $\mu$, there is exactly one $\tau_\mu$ such that $e_{\tau_\mu} w_\mu \ne 0$. As $w_1, \dots, w_m$ annihilate each other, we can consider the multiplication in $\operatorname{lin}\{e_{\tau_\mu}, w_\mu, \dots, w_\mu^{q_\mu}\}$ separately for each $\mu$. This is merely univariate polynomial multiplication. We have already computed the product corresponding to $e_{\tau_\mu} w_\mu^{q_\mu}$ and $w_\mu^{q_\mu} e_{\tau_\mu}$, since $w_\mu^{q_\mu} \in \mathsf{L}_A \cap \mathsf{R}_A$. Thus it is sufficient to compute the product of two elements in $\operatorname{lin}\{e_{\tau_\mu}, w_\mu, \dots, w_\mu^{q_\mu - 1}\}$ up to the term $w_\mu^{q_\mu}$. This can be done with $2q_\mu - 1$ multiplications via evaluation and interpolation. However, we have already computed the product $\xi_{\tau_\mu} \eta_{\tau_\mu}$. (This corresponds to evaluation at zero.) Hence $2q_\mu - 2$ multiplications suffice. The condition $\#k \ge 2N(A) - 2$ ensures that there are enough scalars to perform evaluation and interpolation.

The total number of multiplications is $t + 2\ell + 2(n - \ell - t) = 2 \dim A - t$. This completes the proof. □

**4. Structural insights.** In the proof of the preliminary characterization results over algebraically closed [4] and perfect [5] fields, the existence of a subalgebra $B$ of $A$ with $B \oplus \operatorname{rad} A = A$ and $B \cong A/\operatorname{rad} A$ played a crucial role. Over perfect (and henceforth over algebraically closed) fields, such an algebra always exists by the Wedderburn–Malcev theorem (Theorem 5). Over nonperfect fields, such an algebra need not exist; see [11, Chapter 6, Example 4] for an example.

Instead of the above algebra $B$, we use a slightly larger subalgebra of $A$. This algebra will be induced by a lifting of the idempotents in $A/\operatorname{rad} A$. Let $\bar{.} : A \to A/\operatorname{rad} A$ denote the canonical projection defined by $a \mapsto \bar{a} = a + \operatorname{rad} A$. Let $A_1 \oplus \cdots \oplus A_t$ be the decomposition of $A/\operatorname{rad} A$ into simple factors. We here write the decomposition of $A/\operatorname{rad} A$ in an additive way; that is, we consider the $A_\tau$ as subspaces of $A/\operatorname{rad} A$. This is done to simplify notation, mainly to write $A_\tau$ instead of $\{0\} \times \cdots \times \{0\} \times A_\tau \times \{0\} \times \cdots \times \{0\}$, which would be the corresponding subspace of $A_1 \times \cdots \times A_t$. Let $e_\tau$ denote the identity of $A_\tau$ for $1 \le \tau \le t$. Then $e_1 + \cdots + e_t$ is a *decomposition of the identity* of $A/\operatorname{rad} A$; that is,

1. $1 = e_1 + \cdots + e_t$,
2. the $e_\tau$ are idempotents, i.e., $e_\tau^2 = e_\tau$ for all $\tau$, and
3. the $e_\tau$ annihilate each other, i.e., $e_\sigma e_\tau = 0$ for $\sigma \ne \tau$.

(It is actually more precise to call $e_1, \dots, e_t$ the decomposition of the identity, since $e_1 + \cdots + e_t$ is simply 1. We will, however, use the notation $e_1 + \cdots + e_t$.) Now the crucial point is that this decomposition can be lifted to $A$: By [11, Corollary 3.3.9], there is a corresponding decomposition of the identity $f_1 + \cdots + f_t$ in $A$ (i.e., $f_1, \dots, f_t$ fulfill the three conditions above, where 1 is the identity of $A$) such that

$$e_\tau = \bar{f}_\tau = f_\tau + \operatorname{rad} A \qquad \text{for all } \tau. \tag{9}$$

We remark that the decomposition $e_1 + \cdots + e_t$ is a *central* decomposition of the identity in $A/\operatorname{rad} A$; i.e., $e_\tau a = a e_\tau$ holds for all $\tau$ and for all $a \in A/\operatorname{rad} A$.

The following theorem is useful in what follows. For a proof, see [11, Theorem 3.5.3].

THEOREM 23. *Let $e_1 + \cdots + e_t$ be a central decomposition of the identity in $A/\operatorname{rad} A$. Then $f_i(\operatorname{rad} A)f_j = f_iAf_j$ for all $i \neq j$ and $f_i(\operatorname{rad} A)f_i = \operatorname{rad}(f_iAf_i)$ for all $i$. (Here $f_1 + \cdots + f_t$ is the lifted decomposition in $A$ defined above.)*

The next lemma shows that $B := f_1Af_1 + \cdots + f_tAf_t$ is a subalgebra of $A$ such that $B/\operatorname{rad} B \cong A_1 \oplus \cdots \oplus A_t$. Note that the algebra $B$ always exists; in particular, its existence does not depend on the perfectness of the ground field. The important point is that we have found an algebra with $B/\operatorname{rad} B \cong A_1 \oplus \cdots \oplus A_t = A/\operatorname{rad} A$ for which $B \cap f_iAf_j = \{0\}$ holds for all $i \neq j$. This is needed in the proof of Lemma 31.

LEMMA 24. *Let $A$ be an algebra over an arbitrary field and $A_1 \oplus \cdots \oplus A_t$ be its decomposition into simple algebras. Let $f_1 + \cdots + f_t$ be a decomposition of the identity as in* (9)*. Then the following hold:*

1. *$B := f_1Af_1 + f_2Af_2 + \cdots + f_tAf_t$ is a subalgebra of $A$,*
2. *$B_\tau := f_\tau Af_\tau$ is an algebra for all $\tau$,*
3. *$B_\sigma \cdot B_\tau = \{0\}$ for all $\sigma \neq \tau$,*
4. *$\operatorname{rad} B_\tau = f_\tau(\operatorname{rad} A)f_\tau$ for all $\tau$, and*
5. *$B_\tau/\operatorname{rad} B_\tau \cong A_\tau$ for all $\tau$.*

*Proof.*

1. $B$ is clearly closed under addition and scalar multiplication. $1 \in B$, too. It remains to show that $B$ is also closed under multiplication. This follows from the fact that the $f_\tau$ annihilate each other: We have

$$(f_1a_1f_1 + \cdots + f_ta_tf_t)(f_1b_1f_1 + \cdots + f_tb_tf_t) = f_1a_1f_1b_1f_1 + \cdots + f_ta_tf_tb_tf_t \in B, \tag{10}$$

since $f_\sigma f_\tau = 0$ for $\sigma \neq \tau$.

2. $B_\tau$ is obviously closed under addition, scalar multiplication, and multiplication. Furthermore, $f_\tau$ is the identity of $B_\tau$.
3. We have $f_\sigma af_\sigma \cdot f_\tau bf_\tau = 0$ for all $f_\sigma af_\sigma \in B_\sigma$ and $f_\tau bf_\tau \in B_\tau$, as $f_\sigma \cdot f_\tau = 0$.
4. Since $e_1 + \cdots + e_t$ is a central decomposition, this follows from Theorem 23.
5. We have $A_\tau \cong e_\tau(A/\operatorname{rad} A)e_\tau$, where $e_1 + \cdots + e_t$ is a decomposition of the identity of $A/\operatorname{rad} A$ as in (9). On the other hand, $\operatorname{rad} B_\tau = f_\tau Af_\tau/f_\tau(\operatorname{rad} A)f_\tau$ by the second and fourth statements. We define a mapping $i : A_\tau \to B_\tau/\operatorname{rad} B_\tau$ by $e_\tau(a + \operatorname{rad} A)e_\tau \mapsto f_\tau af_\tau + f_\tau(\operatorname{rad} A)f_\tau$. We claim that $i$ is an isomorphism. First, $i$ is well defined: Consider elements $a$ and $b$ fulfilling $e_\tau(a + \operatorname{rad} A)e_\tau = e_\tau(b + \operatorname{rad} A)e_\tau$. Then by (9),

$$\begin{aligned} f_\tau af_\tau + \operatorname{rad} A &= (f_\tau + \operatorname{rad} A)(a + \operatorname{rad} A)(f_\tau + \operatorname{rad} A) \\ &= (f_\tau + \operatorname{rad} A)(b + \operatorname{rad} A)(f_\tau + \operatorname{rad} A) \\ &= f_\tau bf_\tau + \operatorname{rad} A. \end{aligned} \tag{11}$$

Thus $i$ is well defined.
Obviously, $i(\alpha x) = \alpha i(x)$ and $i(x + y) = i(x) + i(y)$ for all $x, y \in A_\tau$. For $x = e_\tau(a + \operatorname{rad} A)e_\tau$ and $y = e_\tau(b + \operatorname{rad} A)e_\tau$ in $B_\tau$, we have

$$xy = e_\tau(a + \operatorname{rad} A)e_\tau \cdot e_\tau(b + \operatorname{rad} A)e_\tau = e_\tau(af_\tau b + \operatorname{rad} A)e_\tau. \tag{12}$$

Hence,

$$\begin{aligned} i(xy) &= f_\tau af_\tau bf_\tau + f_\tau(\operatorname{rad} A)f_\tau \\ &= (f_\tau af_\tau + f_\tau(\operatorname{rad} A)f_\tau)(f_\tau bf_\tau + f_\tau(\operatorname{rad} A)f_\tau) \\ &= i(x)i(y). \end{aligned}$$

Finally, since $f_\tau a f_\tau \in f_\tau(\operatorname{rad} A)f_\tau$ iff $e_\tau(a + \operatorname{rad} A)e_\tau \subseteq \operatorname{rad} A$, $i$ is also bijective. □

## 5. Main result.

Throughout this section, $A$ denotes a $k$-algebra of minimal rank and $A_1 \oplus \cdots \oplus A_t$ denotes the decomposition of $A/\operatorname{rad} A$ into simple algebras. Again we write the decomposition in an additive way.

By Corollary 9 and Lemma 10, $A/\operatorname{rad} A$ is of minimal rank. By [4, Theorem 2], either $A_\tau \cong k$ or $A_\tau \cong k^{2\times 2}$ or $A_\tau$ is a proper extension field of $k$ for all $\tau$. In the latter case, $\#k \geq 2\dim A_\tau - 2$ also has to hold.

Let $e_\tau$ be the identity of $A_\tau$, and let $f_1 + \cdots + f_t$ be a decomposition of the identity of $A$ as in (9). Consider the *Peirce decomposition* (see [11, p. 26]) of $A$ with respect to $f_1, \dots, f_t$:

$$A = \bigoplus_{1\leq \sigma,\tau \leq t} f_\sigma A f_\tau \qquad \text{and} \qquad \operatorname{rad} A = \bigoplus_{1\leq \sigma,\tau \leq t} f_\sigma (\operatorname{rad} A) f_\tau.$$

Since $f_1, \dots, f_t$ are idempotents that annihilate each other, the pairwise intersections of the $f_\sigma A f_\tau$ equal in fact $\{0\}$.

Compared to the proof over perfect fields, the algebra $f_1 A f_1 \oplus \cdots \oplus f_t A f_t$ now plays the role of the subalgebra $S$ of $A$ with $S \cong A/\operatorname{rad} A = A_1 \oplus \cdots \oplus A_t$ in [5]. The proof in this section is almost the same as that of [5], since we need only the following property (in the proof of Lemma 31): The intersection of each $f_\tau A f_\tau$ with any $f_i(\operatorname{rad} A)f_j$ with $i \neq j$ is the nullspace. Furthermore, the $f_\tau A f_\tau$ annihilate each other like the $A_\tau$ do. The only thing which slightly complicates matters is that instead of knowing that $A_\tau$ is simple, we here have only that $f_\tau A f_\tau / \operatorname{rad} f_\tau A f_\tau$ is simple and in fact equal to $A_\tau$.

### 5.1. A first decomposition step.

If an idempotent, say, $f_1$ fulfils $f_1(\operatorname{rad} A)f_j = f_j(\operatorname{rad} A)f_1 = \{0\}$ for all $j \geq 2$, then the characterization problem splits into two subproblems.

Lemma 25. *If $f_1(\operatorname{rad} A)f_j = f_j(\operatorname{rad} A)f_1 = \{0\}$ for all $2 \leq j \leq t$, then*

$$A \cong f_1 A f_1 \times (f_2 + \cdots + f_t) A (f_2 + \cdots + f_t).$$

*Proof.* Let $f' := f_2 + \cdots + f_t$. First note that $f_1 A f_1$ is an algebra by Lemma 24(2). Along the same lines, it is easy to see that $f' A f'$ is an algebra.

Since $e_\tau = f_\tau + \operatorname{rad} A$ is the identity of $A_\tau$ for all $\tau$, $e_\tau$ is contained in the center of $A/\operatorname{rad} A$ (i.e., $e_\tau a = a e_\tau$ for all $a \in A/\operatorname{rad} A$). Thus by Theorem 23, we have $f_\sigma A f_\tau = f_\sigma(\operatorname{rad} A) f_\tau$ for all $\sigma \neq \tau$. In particular, $f_1 A f_j = f_j A f_1 = \{0\}$ for all $j \geq 2$.

Let $a \in A$ be arbitrary. We can write $a = a_1 + a'$ with unique $a_1 \in f_1 A f_1$ and $a' \in f' A f'$, since $f_1 A f_1 \oplus f' A f' = A$. We define a mapping $A \to f_1 A f_1 \times f' A f'$ by $a \mapsto (a_1, a')$. Exploiting the fact that $f_1 A f' = f' A f_1 = \{0\}$, it is easy to verify that this is an isomorphism of algebras. □

Next we show that if $A$ has minimal rank, both $f_1 A f_1$ and $f' A f'$ have minimal rank. This follows directly from the work of Alder and Strassen.

Lemma 26. *Let $B_1$ and $B_2$ be two algebras. The algebra $B = B_1 \times B_2$ has minimal rank iff both $B_1$ and $B_2$ have minimal rank.*

*Proof.* Assume that $B$ has minimal rank. Let $t_1$, $t_2$, and $t$ be the number of twosided maximal ideals of $B_1$, $B_2$, and $B$, respectively. We have $t = t_1 + t_2$. Note that $\operatorname{rad} B = \operatorname{rad} B_1 \times \operatorname{rad} B_2$. By Corollary 9 and Lemma 10, $R(B) \geq 2\dim \operatorname{rad} B_1 + R((B_1/\operatorname{rad} B_1) \times B_2)$. If we now apply Lemma 11 repeatedly, we obtain $R(B) \geq$

$2\dim B_1 - t_1 + R(B_2)$. Thus $B_2$ has to have minimal rank. $B_1$ is treated completely the same way.

The other direction is trivial. □

Now assume that, say, $A_1$ is either an extension field of dimension at least two or isomorphic to $k^{2\times 2}$. Moreover, let $f_1(\operatorname{rad} A)f_j = f_j(\operatorname{rad} A)f_1 = \{0\}$ for all $2 \le j \le t$. By Lemma 26, the algebra $f_1Af_1$ has to have minimal rank. By Lemma 24(5), we obtain $(f_1Af_1)/\operatorname{rad}(f_1Af_1) = e_1(A/\operatorname{rad} A)e_1 \cong A_1$.

If $A_1$ is an extension field, then $f_1Af_1$ is local and $\dim f_1Af_1/(\operatorname{rad} f_1Af_1) \ge 2$. Such algebras have been characterized by Büchi and Clausen [8]. They are isomorphic to $k[X]/(p(X)^m)$ for some irreducible polynomial $p$ with $\deg p \ge 2$ and some $m \ge 1$ and $\#k \ge 2m \deg p$. The case that $A_1$ is isomorphic to $k^{2\times 2}$ is settled by the following lemma. It turns out that, in this case, we necessarily have $f_1Af_1 \cong k^{2\times 2}$.

LEMMA 27. *Let $C$ be an algebra fulfilling $C/\operatorname{rad} C \cong k^{2\times 2}$. Then $R(C) \ge \frac{5}{2}\dim C - 4$. In particular, if $\operatorname{rad} C \ne \{0\}$, then $C$ is not of minimal rank.*

*Proof.* Let $\beta = (f_1, g_1, w_1, \dots, f_r, g_r, w_r)$ be a bilinear computation for $C$. We may assume w.l.o.g. that $f_1, \dots, f_N$ is a basis of $C^*$, where $N = \dim C$. Let $u_1, \dots, u_N$ be its dual basis. Furthermore, assume that $\bar u_{N-3}, \dots, \bar u_N$ is a basis of $C/\operatorname{rad} C \cong k^{2\times 2}$, where $\bar{\ } : C \to C/\operatorname{rad} C$ denotes the canonical projection. Choose $\alpha_1, \dots, \alpha_4$ such that $1 = \alpha_1\bar u_{N-3} + \dots + \alpha_4 \bar u_N$ (in $C/\operatorname{rad} C$). By Nakayama's lemma (Lemma 4), $\alpha_1 u_{N-3} + \dots + \alpha_4 u_N$ is invertible in $C$. Exploiting sandwiching (see section 2.2), we can achieve that $1 = \alpha_1 u_{N-3} + \dots + \alpha_4 u_N$ in $C$. Choose $\xi_1, \dots, \xi_4$ and $\eta_1, \dots, \eta_4$ such that for $x = \xi_1 u_{N-3} + \dots + \xi_4 u_N$ and $y = \eta_1 u_{N-3} + \dots + \eta_4 u_N$,

$$\bar x = \begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix} \quad \text{and} \quad \bar y = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}.$$

An easy calculation shows that $\bar x\bar y - \bar y \bar x$ is invertible in $C/\operatorname{rad} C$. Once more by Lemma 4, $xy - yx$ is invertible in $C$. Now by [4, Lemma 3.8] (with $m = N - 4$),

$$r \ge 5/2 \dim C - 4. \tag{13}$$

Since $C/\operatorname{rad} C \cong k^{2\times 2}$, the algebra $C/\operatorname{rad} C$ is separable; that is, for any extension field $K$ of $k$, $(C/\operatorname{rad} C) \otimes K \cong K^{2\times 2}$ is semisimple. By the Wedderburn–Malcev theorem (Theorem 5), $C$ contains a subalgebra $C' \cong k^{2\times 2}$ with $C' \oplus \operatorname{rad} C = C$. Hence $\operatorname{rad} C$ is a $k^{2\times 2}$-bimodule. By [4, Fact 8.5], $\operatorname{rad} C \ne \{0\}$ implies $\dim \operatorname{rad} C \ge 4$. Now we can rewrite (13) as

$$r \ge 5/2 \dim C - 4 = 2\dim C + \underbrace{1/2(4 + \dim \operatorname{rad} C) - 4}_{\ge 0 \text{ if } \operatorname{rad} C \ne \{0\}}.$$

This proves the second statement. □

By Theorem 23, $\operatorname{rad}(f'Af') = f'(\operatorname{rad} A)f'$ holds. From this, it follows that $(f'Af')/\operatorname{rad}(f'Af') = e'(A/\operatorname{rad} A)e' = A_2 \oplus \dots \oplus A_t$, as in the proof of Lemma 24(5), where $e' = e_2 + \dots + e_t$. Hence we can proceed recursively with $f'Af'$. The following lemma summarizes these considerations.

LEMMA 28. *Let $A$ be an algebra of minimal rank over an arbitrary field $k$. Then*

$$A \cong C_1 \times \dots \times C_s \times \underbrace{k^{2\times 2} \times \dots \times k^{2\times 2}}_{u \text{ times}} \times B,$$

*where*

1. *the $C_\sigma$ are local algebras of minimal rank with* $\dim C_\sigma/\operatorname{rad} C_\sigma \geq 2$ *and*
2. *$B$ is an algebra of minimal rank.*

*Furthermore, if $B_1 \oplus \cdots \oplus B_r$ is a decomposition of $B/\operatorname{rad} B$ into simple factors, $e_\rho$ denotes the identity of $B_\rho$ for $1 \leq \rho \leq r$, and $f_1 + \cdots + f_r$ is a decomposition of the identity of $B$ as in* (9)*, then for all $B_\rho$ such that $B_\rho$ is either isomorphic to $k^{2\times 2}$ or a proper extension field of $k$ there is a $j_\rho \neq \rho$ such that $f_\rho(\operatorname{rad} B)f_{j_\rho} \neq \{0\}$ or $f_{j_\rho}(\operatorname{rad} B)f_\rho \neq \{0\}$. Above, $s$ and $u$ may be zero and the factor $B$ is optional.*

**5.2. *B* is superbasic.** Next we show that the algebra $B$ in Lemma 28 is always superbasic; that is, all the $B_\rho$ in the decomposition of $B/\operatorname{rad} B$ into simple algebras are isomorphic to $k$. Thereafter, we apply Theorem 22 and obtain our characterization result.

To do so, under the assumption that one of the factors $B_\rho$ is not isomorphic to $k$, we construct some algebras that have to have minimal rank because $B$ has minimal rank. Then we prove that none of the constructed algebras has minimal rank, obtaining a contradiction.

The next lemma shows that it suffices to consider the problem modulo $(\operatorname{rad} B)^2$ in the sense that, modulo $(\operatorname{rad} B)^2$, the algebra $B/(\operatorname{rad} B)^2$ still has the same properties as $B$ in Lemma 28. For any algebra $C$, let $\tilde{ } : C \to C/(\operatorname{rad} C)^2$ denote the canonical projection defined by $a \mapsto \tilde{a} = a + (\operatorname{rad} C)^2$.

LEMMA 29. *Let $A$ be an algebra of minimal rank over an arbitrary field $k$. Let $A \cong C_1 \times \cdots \times C_s \times k^{2\times 2} \times \cdots \times k^{2\times 2} \times B$, as in Lemma 28, and let $\tilde{B}$ and $\tilde{B}_1, \ldots, \tilde{B}_r$ be the images of $B$ and $B_1, \ldots, B_r$ under the canonical projection $B \to B/(\operatorname{rad} B)^2$. Then $\tilde{B}$ has minimal rank, and for all $\tilde{B}_\rho$ such that $\tilde{B}_\rho$ is either isomorphic to $k^{2\times 2}$ or a proper extension field of $k$, there is a $j'_\rho \neq \rho$ such that $\tilde{f}_\rho(\operatorname{rad} \tilde{B})\tilde{f}_{j'_\rho} \neq \{0\}$ or $\tilde{f}_{j'_\rho}(\operatorname{rad} \tilde{B})\tilde{f}_\rho \neq \{0\}$.*

*Proof.* If $B$ has minimal rank, so has $\tilde{B}$, by Corollary 9. Since for each $B_\rho$ that is either isomorphic to $k^{2\times 2}$ or a proper extension field of $k$, there is a $j_\rho \neq \rho$ such that $f_\rho(\operatorname{rad} B)f_{j_\rho} \neq \{0\}$ or $f_{j_\rho}(\operatorname{rad} B)f_\rho \neq \{0\}$, the next lemma shows the existence of the index $j'_\rho$. ◻

LEMMA 30. *Let $C$ be an algebra and $C_1 \oplus \cdots \oplus C_r$ be the decomposition of $C/\operatorname{rad} C$ into simple factors. Let $e_\rho$ be the identity of $C_\rho$ for all $\rho$, and let $f_1 + \cdots + f_r$ be a decomposition of the identity as in* (9)*. For all indices $i$ and $j$ with $i \neq j$ such that $f_i(\operatorname{rad} C)f_j \neq \{0\}$, there is an index $j'$ with $i \neq j'$ such that $\tilde{f}_i(\operatorname{rad} \tilde{C})\tilde{f}_{j'} \neq \{0\}$. In the same way, there is an index $i'$ with $i' \neq j$ such that $\tilde{f}_{i'}(\operatorname{rad} \tilde{C})\tilde{f}_j \neq \{0\}$.*

*Proof.* Let $x$ be such that $f_i x f_j \neq 0$ and $f_i x f_j \in f_i(\operatorname{rad} C)f_j$. Since $f_i C f_j = f_i(\operatorname{rad} C)f_j$ by Theorem 23, we may assume $x \in \operatorname{rad} C$. Let $n \geq 1$ be the unique number such that $x \in (\operatorname{rad} C)^n \setminus (\operatorname{rad} C)^{n+1}$. By definition, there are $x_1, \ldots, x_n \in \operatorname{rad} C \setminus (\operatorname{rad} C)^2$ such that $x = x_1 \cdots x_n$. Since $1 = f_1 + \cdots + f_r$,

$$f_i x_1(f_1 + \cdots + f_r)x_2(f_1 + \cdots + f_r) \cdots (f_1 + \cdots + f_r)x_n f_j = f_i x f_j \neq 0.$$

Thus there are indices $h_1, \ldots, h_{n-1}$ such that

$$f_i x_1 f_{h_1} x_2 f_{h_2} \cdots f_{h_{n-1}} x_n f_j \neq 0.$$

If all $h_\nu$ equal $i$, then $f_{h_{n-1}} x_n f_j = f_i x_n f_j \neq 0$, and we set $j' = j$. Since $f_i x_n f_{j'} \in \operatorname{rad} C \setminus (\operatorname{rad} C)^2$, $\tilde{f}_i \tilde{x}_n \tilde{f}_{j'} \neq 0$, and hence we are done. Otherwise let $\nu_0$ be the smallest index such that $h_{\nu_0} \neq i$. Now we set $j' = h_{\nu_0}$ and can conclude in the same way as before that $\tilde{f}_i(\operatorname{rad} \tilde{C})\tilde{f}_{j'} \neq \{0\}$.

The index $i'$ is constructed in the identical way. ∎

W.l.o.g. we assume from now on that $(\operatorname{rad} B)^2 = \{0\}$. $B_1$ is either isomorphic to $k^{2\times 2}$ or a (not necessarily proper) extension field of $k$. We first construct two algebras, which have to have minimal rank since $B$ has minimal rank. Next we prove that these algebras cannot have minimal rank if $B_1$ is isomorphic to $k^{2\times 2}$ or a proper extension field of $k$. Hence $B_1 \cong k$ must hold. By symmetry, this has to hold for all $B_1, \dots, B_r$. It follows that $B$ is superbasic.

The next lemma shows how to construct these two algebras. By symmetry, the lemma also holds for $(\operatorname{rad} B) f_1 \neq \{0\}$.

LEMMA 31. *If $f_1(\operatorname{rad} B) \neq \{0\}$, then there is an index $2 \leq j \leq s$ and a nonzero $(B_1, B_j)$-bimodule $M$ such that the algebra $B_1 \times B_j \times M$ (as vector spaces) with multiplication law $(a, b, x) \cdot (a', b', x') = (aa', bb', ax' + xb')$ has minimal rank.*

*Proof.* We decompose $\operatorname{rad} B$ as

$$\operatorname{rad} B = \bigoplus_{1 \leq \rho, \eta \leq r} f_\rho (\operatorname{rad} B) f_\eta. \tag{14}$$

By Lemma 29, there is an index $j \geq 2$ such that $f_1(\operatorname{rad} B) f_j \neq \{0\}$. W.l.o.g. we may assume that $j = 2$.

Consider $C := f_1 B f_1 + f_2 B f_2$. As in Lemma 24(1), it is easy to see that $C$ is a subalgebra of $B$. As in Lemma 24(4), $\operatorname{rad} C = f_1(\operatorname{rad} B) f_1 + f_2(\operatorname{rad} B) f_2$. Finally $C/\operatorname{rad} C \cong B_1 \oplus B_2$. Since the pairwise intersection of the $f_\rho(\operatorname{rad} B) f_\eta$ in (14) are the nullspace, we also have $C/\operatorname{rad} B \cong B_1 \oplus B_2$.

Let

$$I = \bigoplus_{(\rho,\eta) \neq (1,2)} f_\rho (\operatorname{rad} B) f_\eta.$$

$I$ is a twosided ideal of $B$ (see also [4, p. 105]). Since $B$ is of minimal rank and $I \subseteq \operatorname{rad} B$, $B/I$ is also of minimal rank by Corollary 9 and Lemma 10. By [11, Corollary 3.1.14], we have $\operatorname{rad}(B/I) = (\operatorname{rad} B)/I \cong f_1(\operatorname{rad} B) f_2 / I \neq \{0\}$. Let $\hat{\ } : B \to B/I$ denote the canonical projection defined by $a \mapsto \hat{a} = a + I$. Since $I \subseteq \operatorname{rad} B$, $\hat{f}_1, \dots, \hat{f}_r$ are idempotent elements that annihilate each other, because this also holds for $f_1, \dots, f_r$. Let $\hat{f} = \hat{f}_3 + \cdots + \hat{f}_r$. Then for all $a \in \hat{f}(B/I)\hat{f}$, we have $a(\operatorname{rad} B/I) = (\operatorname{rad} B/I)a = \{0\}$, since the $\hat{f}_\rho$ annihilate each other. This means that the elements in $(\hat{f}_1 + \hat{f}_2)(B/I)(\hat{f}_1 + \hat{f}_2)$ annihilate the elements in $\hat{f}(B/I)\hat{f}$ and vice versa. Then $B/I$ is isomorphic to the product of these two algebras (see, e.g., [4, Lemma 8.3] for a proof), i.e.,

$$B/I \cong (\hat{f}_1 + \hat{f}_2)\,(B/I)\,(\hat{f}_1 + \hat{f}_2) \times \hat{f}\,(B/I)\,\hat{f}.$$

By Lemma 26, $B/I$ is of minimal rank only if $D := (\hat{f}_1 + \hat{f}_2)(B/I)(\hat{f}_1 + \hat{f}_2)$ is of minimal rank.

Because $C := f_1 B f_1 + f_2 B f_2$ is a subalgebra of $(f_1 + f_2) B (f_1 + f_2)$, $\hat{C}$ is a subalgebra of $D$. This subalgebra $\hat{C}$ is isomorphic to $B_1 \oplus B_2$, since $\operatorname{rad} C = f_1(\operatorname{rad} B) f_1 + f_2(\operatorname{rad} B) f_2$. Hence $D$ has a subalgebra that is isomorphic to $B_1 \oplus B_2$. The radical of $D$ is $\operatorname{rad} D \cong f_1(\operatorname{rad} B) f_2 / I \neq \{0\}$.

Noting that $\operatorname{rad} D$ is a $(B_1, B_2)$-bimodule that fulfills the multiplication law in the lemma completes the proof. ∎

The two lemmas below show that the algebras constructed in Lemma 31 cannot have minimal rank. Hence we have obtained the following result.

Lemma 32. *The algebra $B$ in Lemma 28 is superbasic.*

We have two possible choices for $B_1$ in Lemma 31, namely, $B_1 \cong k^{2\times 2}$ and $B_1$ is a proper extension field of $k$. We split them into several subcases depending on $B_2$. The cases $B_1 \cong k^{2\times 2}$ and $B_2$ isomorphic to either $k$ or to $k^{2\times 2}$ are treated in [4, Lemmas 8.7, 8.8]. Lemma 33 below treats the case where $B_1$ is a proper extension field of $k$ and $B_2$ is either also a proper extension field of $k$ or $k$ itself. Finally, Lemma 34 settles the case where $B_1 \cong k^{2\times 2}$ and $B_2$ is a proper extension field. By symmetry this also settles the remaining case where the roles of $B_1$ and $B_2$ are interchanged.

Lemma 33. *Let $K$ be a proper extension field of $k$ and $L$ be a (not necessarily proper) extension field of $k$. Let $M$ be a nonzero $(K, L)$-bimodule, and define the algebra $A = K \times L \times M$ (as vector spaces) with multiplication law $(a, b, x)\cdot(a', b', x') = (aa', bb', ax' + xb')$. Then*

$$R(A) \geq 2\dim L + \dim M + (2\dim K - 1)\frac{\dim M}{\dim K + 1} - 1.$$

*In particular, $A$ is not of minimal rank.*

*Proof.* Let $\beta = (f_1, g_1, w_1, \ldots, f_r, g_r, w_r)$ be an optimal computation for $A$, and let $n = \dim A$. We can achieve that $w_1, \ldots, w_n$ form a basis of $A$. W.l.o.g. we may assume that $\mathrm{lin}\{w_1, \ldots, w_\ell\} + (K \times \{0\} \times M) = A$, where $\ell = \dim L$. Let $W_1 = \mathrm{lin}\{w_1, \ldots, w_{\ell-1}\}$. By definition, $\beta$ separates the triple $(\{0\}, \{0\}, W_1)$.

Next we show that the computation $\beta$ also separates the triple $(\{0\} \times \{0\} \times M, \{0\}, W_1)$. If this were not the case, then there would be a nonzero $a \in \{0\}\times\{0\}\times M$ by Lemma 12 such that

$$a \cdot A \subseteq (\{0\} \times \{0\} \times M) \cdot \{0\} + W_1 = W_1,$$

which cannot be the case, as $W_1 \cap (\{0\} \times \{0\} \times M) = \{0\}$.

We claim that $\beta$ even separates $(\{0\} \times L \times M, \{0\}, W_1)$. Otherwise, Lemma 12 implies that there would be some $a \in (\{0\} \times L \times M) \setminus (\{0\} \times \{0\} \times M)$ such that

$$a \cdot A \subseteq (\{0\} \times L \times M) \cdot \{0\} + W_1 = W_1.$$

As $a \in (\{0\} \times L \times M) \setminus (\{0\} \times \{0\} \times M)$, we have $\dim a \cdot A \geq \dim L = \ell$, contradicting $\dim W_1 \leq \ell - 1$.

Define $\beta' = (f_1, g_1', w_1, \ldots, f_r, g_r', w_r)$ through $g_\rho' = g_\rho'|_{K\times\{0\}\times M}$ for all $\rho$. From the definition of "separate," it follows that $\beta'$ also separates $(\{0\} \times L \times M, \{0\}, W_1)$.

Let $\phi$ be the multiplication map of $A$. Then $\beta'$ is a computation for the bilinear map $\psi := \phi|_{A\times(K\times\{0\}\times M)}$, which we can view as the multiplication $a \cdot (a', x') \mapsto (aa', ax')$ of the $K$-left module $K \times M$. Let $\pi$ be a projection of $A$ onto $K \times \{0\} \times M$ with $W_1 \subseteq \ker \pi$. Since $\mathrm{im}\, \psi \subseteq K \times \{0\} \times M$, $\pi \circ \psi = \psi$.

Now by Lemma 8

$$\begin{aligned} R(A) &\geq R\big(\psi/((\{0\} \times L \times M) \times \{0\})\big) + \dim L + \dim M + \#W_1 \\ &\geq R\big(\psi/((\{0\} \times L \times M) \times \{0\})\big) + 2\dim L + \dim M - 1. \end{aligned} \tag{15}$$

Since we have $\psi(\{0\} \times L \times M, K \times \{0\} \times M) = \{0\}$, we can still view $\psi/((\{0\} \times L \times M) \times \{0\})$ as the multiplication of the $K$-left module $K \times M$. By Wedderburn's theorem for modules [19], $K \times M \cong K^m$, where $m = \dim(K \times M)/\dim K$. (It also follows that $m$ is necessarily an integer strictly greater than one.) By Hartmann's lower bounds for the rank of modules [17, Theorem 2],

$$R\big(\psi/((\{0\} \times L \times M) \times \{0\})\big) \geq (2\dim K - 1) \cdot \frac{\dim M}{\dim K + 1}.$$

This proves the first claim of the lemma.

To see that the last bound implies that $A$ is not of minimal rank, we rewrite the bound as

$$R(A) \geq 2\dim L + 3\dim M + 2\dim K - \frac{\dim M}{\dim K} - 2.$$

Since $A$ has two maximal ideals, the second claim of the lemma is proven if $\dim M - \dim M/\dim K - 1 \geq 0$. But this is clearly true, because $\dim K \geq 2$ and $\dim M \geq \dim K \geq 2$. □

LEMMA 34. *Let $L$ be a proper extension field of $k$, and let $M$ be a nonzero $(k^{2\times 2}, L)$-bimodule. Define the algebra $A = k^{2\times 2} \times L \times M$ (as vector spaces) with multiplication law $(a, b, x) \cdot (a', b', x') = (aa', bb', ax' + xb')$. Then*

$$R(A) \geq 2\dim L + \frac{5}{2}\dim M + 6.$$

*In particular, $A$ is not of minimal rank.*

*Proof.* Let $\phi$ denote the multiplication of $A$, and define the bilinear map $\psi := \phi|_{A\times(k^{2\times 2}\times\{0\}\times M)}$. The same reasoning as in Lemma 33 shows that

$$R(A) \geq R\big(\psi/((\{0\} \times L \times M) \times \{0\})\big) + 2\dim L + \dim M - 1,$$

since we did not make use of fact that $K$ is an extension field in the proof of (15) in Lemma 33.

Since $(\{0\} \times L \times M) \cdot (k^{2\times 2} \times \{0\} \times M) = \{0\}$, $\psi/((\{0\} \times L \times M) \times \{0\})$ is the multiplication of the $k^{2\times 2}$-module $k^{2\times 2} \times M$. By Wedderburn's theorem for modules, this multiplication corresponds to the multiplication of $2 \times 2$-matrices with $2 \times m$-matrices, where $m = \frac{1}{2}(\dim k^{2\times 2} \times M) = 2 + \frac{1}{2}\dim M$. Since the rank of the multiplication of $2 \times 2$-matrices with $2 \times m$-matrices has the lower bound $3m + 1$ (see [7]), we obtain

$$R(A) \geq 2\dim L + \frac{5}{2}\dim M + 6.$$

To see that this implies that the algebra $A$ does not have minimal rank, note that $\dim A = 4 + \dim L + \dim M$, $A$ has two maximal ideals, and $\frac{1}{2}\dim M > 0$. □

**5.3. The final proof.** By Lemma 28, $A$ is isomorphic to

$$A \cong C_1 \times \cdots \times C_s \times \underbrace{k^{2\times 2} \times \cdots \times k^{2\times 2}}_{u \text{ times}} \times B,$$

where the $C_\sigma$ are local algebras of minimal rank with $\dim C_\sigma/\operatorname{rad} C_\sigma \geq 2$. By Lemma 32, $B$ is a superbasic algebra of minimal rank. By Theorem 22, we obtain the desired characterization of algebras of minimal rank over arbitrary fields.

THEOREM 35. *An algebra $A$ over an arbitrary field $k$ is an algebra of minimal rank iff*

$$A \cong C_1 \times \cdots \times C_s \times \underbrace{k^{2\times 2} \times \cdots \times k^{2\times 2}}_{u \text{ times}} \times B, \tag{16}$$

*where $C_1, \ldots, C_s$ are local algebras of minimal rank with $\dim(C_\sigma/\operatorname{rad} C_\sigma) \geq 2$, i.e., $C_\sigma \cong k[X]/(p_\sigma(X)^{d_\sigma})$ for some irreducible polynomial $p_\sigma$ with $\deg p_\sigma \geq 2$, $d_\sigma \geq 1$,* *and* $\#k \geq 2\dim C_\sigma - 2$, *and* $B$ *is a superbasic algebra of minimal rank; that is, there exist* $w_1, \dots, w_m \in \operatorname{rad} B$ *with* $w_i^2 \neq 0$ *and* $w_i w_j = 0$ *for* $i \neq j$ *such that*

$$\operatorname{rad} B = \mathsf{L}_B + Bw_1B + \cdots + Bw_mB = \mathsf{R}_B + Bw_1B + \cdots + Bw_mB$$

*and* $\#k \geq 2N(B) - 2$. *Any of the integers* $s$, $u$, *or* $m$ *may be zero, and the factor* $B$ *in* (16) *is optional.*

## REFERENCES

[1] A. Alder and V. Strassen, *On the algorithmic complexity of associative algebras*, Theoret. Comput. Sci., 15 (1981), pp. 201–211.

[2] A. Averbuch, Z. Galil, and S. Winograd, *Classification of all minimal bilinear algorithms for computing the coefficients of the product of two polynomials modulo a polynomial, Part* II*: The algebra* $G[u]/\langle u^n \rangle$, Theoret. Comput. Sci., 86 (1991), pp. 143–203.

[3] M. Bläser, *Lower bounds for the multiplicative complexity of matrix multiplication*, Comput. Complexity, 8 (1999), pp. 203–226.

[4] M. Bläser, *Lower bounds for the bilinear complexity of associative algebras*, Comput. Complexity, 9 (2000), pp. 73–112.

[5] M. Bläser, *Algebras of minimal rank over perfect fields*, in Proceedings of the 17th Annual IEEE Computational Complexity Conference (CCC), Montreal, 2002, IEEE Press, Piscataway, NJ, pp. 113–122.

[6] M. Bläser, *Algebras of minimal rank over arbitrary fields*, in Proceedings of the 19th Annual Symposium on Theoretical Aspects of Computer Science (STACS), Berlin, 2003, Lecture Notes in Comput. Sci. 2607, Springer-Verlag, New York, 2003, pp. 403–414.

[7] R. W. Brockett and D. Dobkin, *On the optimal evaluation of a set of bilinear forms*, Linear Algebra Appl., 19 (1978), pp. 207–235.

[8] W. Büchi and M. Clausen, *On a class of primary algebras of minimal rank*, Linear Algebra Appl., 69 (1985), pp. 249–268.

[9] P. Bürgisser, M. Clausen, and M. A. Shokrollahi, *Algebraic Complexity Theory*, Springer, New York, 1997.

[10] P. M. Cohn, *Algebra*, Vol. 3, Wiley, New York, 1991.

[11] Y. A. Drozd and V. V. Kirichenko, *Finite Dimensional Algebras*, Springer, New York, 1994.

[12] W. Everly and M. Giesbrecht, *Efficient decomposition of separable algebras*, J. Symbolic Comput., 37 (2004), pp. 35–81.

[13] K. Friedl and L. Rónyai, *Polynomial time solutions of some problems in abstract algebra*, in Proceedings of the 17th Annual ACM Symposium on Theory of Computing (STOC), Providence, RI, 1985, ACM, New York, 1985, pp. 153–162.

[14] H. F. de Groote, *Characterization of division algebras of minimal rank and the structure of their algorithm varieties*, SIAM J. Comput., 12 (1983), pp. 101–117.

[15] H. F. de Groote, *Lectures on the Complexity of Bilinear Problems*, Lecture Notes in Comput. Sci. 245, Springer, New York, 1986.

[16] H. F. de Groote and J. Heintz, *Commutative algebras of minimal rank*, Linear Algebra Appl., 55 (1983), pp. 37–68.

[17] W. Hartmann, *On the multiplicative complexity of modules over associative algebras*, SIAM J. Comput., 14 (1985), pp. 383–395.

[18] J. Heintz and J. Morgenstern, *On associative algebras of minimal rank*, in Proceedings of the 2nd Applied Algebra and Error Correcting Codes Conference (AAECC), Lecture Notes in Comput. Sci. 228, Springer, New York, 1986, pp. 1–24.

[19] R. S. Pierce, *Associative Algebras*, Springer, New York, 1982.

[20] V. Strassen, *Gaussian elimination is not optimal*, Numer. Math., 13 (1969), pp. 354–356.

[21] V. Strassen, *Algebraic complexity theory*, in Handbook of Theoretical Computer Science, Vol. A, J. van Leeuwen, ed., Elsevier Science, New York, 1990, pp. 634–672.

[22] S. Winograd, *On multiplication in algebraic extension fields*, Theoret. Comput. Sci., 8 (1979), pp. 359–377.

SIAM J. COMPUT.
Vol. 34, No. 2, pp. 299–317

# NONINDEPENDENT RANDOMIZED ROUNDING AND AN APPLICATION TO DIGITAL HALFTONING*

BENJAMIN DOERR†

**Abstract.** We investigate the problem of rounding a given $[0,1]$-valued matrix to a $0,1$ matrix such that the rounding error with respect to $2 \times 2$ boxes is small. Such roundings yield good solutions for the digital halftoning problem, as shown by Asano et al. [*Proceedings of the* 13*th Annual ACM-SIAM Symposium on Discrete Algorithms*, San Francisco, 2002, SIAM, Philadelphia, 2002, pp. 896–904]. We present a randomized algorithm computing roundings with expected error at most 0.5463 per box, improving the 0.75 nonconstructive bound of Asano et al. Our algorithm is the first to solve this problem fast enough for practical application, namely, in linear time.

Of broader interest might be our rounding scheme, which is a modification of randomized rounding. Instead of independently rounding the variables, we impose a number of suitable dependencies. Thus, by equipping the rounding process with some of the problem information, we reduce the rounding error significantly compared to independent randomized rounding, which leads to an expected error of 0.82944 per box. Finally, we give a characterization of realizable dependencies.

**Key words.** randomized rounding, discrepancy, digital halftoning

**AMS subject classifications.** 68W20, 68W25, 11K38, 68R05

**DOI.** 10.1137/S0097539703430154

## 1. Introduction.

In this paper, we are concerned with rounding problems. In their general form, these problems are of the following type: Given some numbers $x_1, \dots, x_n$, one is looking for roundings $y_1, \dots, y_n$ such that some given error measures are small. By rounding we always mean that $y_i = \lfloor x_i \rfloor$ or $y_i = \lceil x_i \rceil$. Since there are $2^n$ possibilities, such rounding problems are good candidates for hard problems. In fact, even several restricted versions like the discrepancy problem are known to be NP-hard. An example related to the problems considered in this paper can be found in Asano, Matsui, and Tokuyama [3, 4].

On the other hand, there are cases that can be solved optimally in polynomial time. Knuth [17], for example, has shown that there exists a rounding such that the errors $|\sum_{i=1}^{k}(y_i - x_i)|$ and $|\sum_{i=1}^{k}(y_{\pi(i)} - x_{\pi(i)})|$ for a fixed permutation $\pi$ and all $1 \le k \le n$ are at most $\frac{n}{n+1}$. Such roundings can be obtained by computing a maximum flow in a network. A recent generalization [7] shows this bound for arbitrary totally unimodular rounding problems.

### 1.1. The digital halftoning problem: A matrix rounding problem.

The rounding problem considered in this paper is motivated by an application from image processing. The *digital halftoning problem* is to convert a continuous-tone intensity image (each pixel may have an arbitrary "color" on the white-to-black scale) into a binary image (only black and white dots are allowed). An intensity image can be represented by a $[0,1]$-valued $m \times n$ matrix $A$. Each entry $a_{ij}$ corresponds to the brightness level of the pixel with coordinates $(i,j)$. Since many devices, e.g., laser

*Received by the editors June 24, 2003; accepted for publication (in revised form) June 8, 2004; published electronically December 1, 2004. A preliminary version of this work appeared in the Proceedings of ESA 2002 [8] and SODA 2003 [9]. This work was partially supported (associate member) by the graduate school "Effiziente Algorithmen und Multiskalenmethoden," Deutsche Forschungsgemeinschaft.

http://www.siam.org/journals/sicomp/34-2/43015.html

†Mathematisches Seminar II, Christian–Albrechts–Universität zu Kiel, 24098 Kiel, Germany (bed@numerik.uni-kiel.de).

printers, can output only white and black dots, we have to round $A$ towards a $0, 1$ matrix. Naturally, this has to be done in such a way that the resulting image looks similar to the original.

This notion of similarity is a crucial point. From the viewpoint of application, similarity is defined via the human visual system: A rounding is good if an average human being can retrieve most of the original information from the rounded image. Using this "criterion," several algorithms turned out to be useful. Floyd and Steinberg [11] proposed the error diffusion algorithm, which rounds the entries one by one and distributes the rounding error over neighboring not-yet-rounded entries. Lippel and Kurland [21] and Bayer [5] investigated the ordered dither algorithm, which partitions the image into fixed size submatrices and rounds each submatrix by comparing its entries with a threshold matrix of the same size. One advantage of this approach is that it can be parallelized easily. Knuth [16] combined ideas from both approaches to get an algorithm called dot diffusion.

Though some work has been done in this direction, less understanding seems to be present on the theoretical side. In particular, a good mathematical formulation of similarity seems hard to find. Such a similarity measure is desirable for two reasons. First, it would allow us to compare algorithms without extensive experimental testing. This is particularly interesting, since comparing different halftonings is a delicate issue. For example, it makes a huge difference whether the images are viewed on a computer screen or are printed on a laser printer. Different printers can also give different impressions. Therefore, a more objective criterion would be very helpful. A second reason is that, in attaining a good criterion, one would have a clearer indication of how a digital halftoning algorithm should work. Thus developing good algorithms would be easier.

So far, the most widely accepted criterion for a good halftoning algorithm is that it has the "blue noise" property (first detected by Ulichney [34]; cf. also the surveys Ulichney [33] and Lau and Arce [19]). This refers not to a similarity measure comparing two images, but an analysis of how the algorithm performs on constant grey level areas. Thus, on the other hand, it gives no information on how changing intensities, in particular shapes, are reproduced. This work has been extended by Sullivan, Ray, and Miller [32], who use a model based on the human visual system to compute good halftoning patterns for constant intensity areas.

Significant research has been done on "global" (regarding the whole image) similarity measures; see, e.g., Lieberman and Allebach [20] and the references therein. However, in many cases the complexity of these measures makes a theoretical investigation almost impossible. Even computing approximations efficiently is often difficult.

At the 2002 international ACM/SIAM Symposium on Discrete Algorithms, Asano et al. [2] presented a structurally much simpler similarity measure, together with several theoretical results. Their experimental studies indicate that good digital halftonings have small error with respect to all $2 \times 2$ subregions. This yields the following problem.

**1.2. Problem statement and results.** Let $A \in [0,1]^{m \times n}$ denote our input matrix. The set $R_{ij} := \{i, i+1\} \times \{j, j+1\} = \{(i,j), (i+1,j), (i,j+1), (i+1,j+1)\}$ for some $i \in [m-1], j \in [n-1]$, is called a $2 \times 2$ subregion (or box) in $[m] \times [n]$.[1] Denote by $\mathcal{R}$ the set of all these boxes. We write $A_{R_{ij}}$ for the $2 \times 2$ matrix $\left(\begin{smallmatrix} a_{i,j} & a_{i,j+1} \\ a_{i+1,j} & a_{i+1,j+1} \end{smallmatrix}\right)$

[1]For an arbitrary number $r$ we denote by $[r]$ the set of positive integers not exceeding $r$.

induced by $R_{ij}$. For any matrix $A$ set $\Sigma A := \sum_{i,j} a_{ij}$, the sum of all its entries.

For a matrix $B \in \{0,1\}^{m \times n}$—which is a rounding of $A$ unless we have $a_{ij}, b_{ij} \in \mathbb{Z}$ and $a_{ij} \neq b_{ij}$ for some $i, j$—we define the rounding error of $A$ with respect to $B$ by

$$d_{\mathcal{R}}(A,B) := \sum_{R \in \mathcal{R}} |\Sigma A_R - \Sigma B_R| \,.$$

We usually omit the subscript $\mathcal{R}$ when there is no danger of confusion.

Asano et al. [2] demonstrated that roundings $B$ such that $d(A,B)$ is small yield good digital halftonings. They showed that for any $A$ an optimal rounding $B^*$ satisfies $d(A,B^*) \leq 0.75|\mathcal{R}|$. They also gave a polynomial time algorithm computing a rounding $B$ such that $d(A,B) - d(A,B^*) \leq 0.5625|\mathcal{R}|$. It is easy to see that there are matrices $A$ such that all roundings (and in fact all integral matrices) $B$ have $d(A,B) \geq 0.5|\mathcal{R}|$.

A major drawback of the algorithm given in [2] is that it is not very practical, as it requires the solution of an integer linear program with totally unimodular constraint matrix. This leads to a run-time bound that is at least quadratic in the number $nm$ of pixels. As pointed out in [2], this is too slow for a real world application.

In this paper, we present a randomized algorithm that runs in linear time. It may be implemented in parallel without problems. This divides the run-time by the number of processors available. The roundings computed by our algorithm have an expected error that exceeds the optimal one by at most $0.3125|\mathcal{R}|$ (instead of $0.5625|\mathcal{R}|$). They also satisfy the absolute bound $E(d(A,B)) \leq 0.5463|\mathcal{R}|$, beating the nonalgorithmic bound of $0.75|\mathcal{R}|$ in [2].

The distribution of the rounding error resulting from this algorithm is highly concentrated around the expected value. The probability that the error exceeds the expected one by more than $\varepsilon|\mathcal{R}|$ is bounded by $\exp(-\Omega(\varepsilon^2\sqrt{|\mathcal{R}|}))$. Our algorithm can also be derandomized. This yields a deterministic linear time algorithm computing roundings with an error guarantee at most equal to that of the randomized version.

For an experimental study of how well our algorithm performs in practice, we refer to Schnieder [25]. Some of these results are also contained in [9].

**1.3. Nonindependent randomized rounding.** The key idea of our algorithm might also be of broader interest. We develop a randomized rounding scheme where the individual roundings are not independent. The classical approach of randomized rounding due to Raghavan and Thompson [23] and Raghavan [22] is to round each variable independently with probability depending on the fractional part of its value. This allows the use of Chernoff-type large deviation inequalities, showing that a sum of independent random variables is highly concentrated around its expectation.

Randomized rounding has been applied to numerous combinatorial optimization problems that can be formulated as integer linear programs (cf. Srinivasan [28]). Though very effective in the general case, a known difficulty with randomized rounding is how to use structural information about the underlying problem. Since only the solution of the linear relaxation is used, further information about the underlying problem cannot be exploited. One idea to overcome this is to use correlation among the events in the analysis of randomized rounding. This allows us to strengthen the classical bounds for packing and covering problems, as shown by Srinivasan [29].

In this paper, we try so use such structure in an earlier phase, namely, in the design of the random experiment. This leads to randomized roundings where the variables are not rounded independently. There have been few attempts to use nonindependent roundings. A straightforward one already appeared in the work of Raghavan and

Thompson and is highly motivated by the structure of the problem. In many cases, the linear program under consideration contains constraints of the kind

$$x_{i_1} + \cdots + x_{i_k} = 1. \tag{1}$$

Typical examples are the vector selection problem (exactly one vector has to be chosen from each of the given sets; cf. Raghavan [22]) or multicommodity flow problems (exactly one path has to be chosen from a nonintegral mixture of paths adding up to the flow; cf. Raghavan and Thompson [23]). In these cases, the probability that an independent randomized rounding is feasible, i.e., that $y_{i_1} + \cdots + y_{i_k} = 1$, can be arbitrarily close to $\frac{1}{e}$. If the number of equations of type (1) is large, this yields an exponentially small success probability.

The solution is to pick one of the variables $x_{i_1}, \ldots, x_{i_k}$ with probability given by its value in the relaxation, and set this variable to one and all others to zero. Thus the rounding is not done independently, but subject to the constraint that exactly one variable receives the value of one. A recent work of Srinivasan [30] extend this to constraints where sums of variables are required to have a particular value other than one.

Another paper on nonindependent roundings is Bertsimas, Teo, and Vohra [6]. They use dependent roundings to give alternative proofs of integrality for several classical polyhedra. They also use "global" dependencies that impose restrictions on the number of variables rounded up or down. For the integrality gap of MINSAT with clauses having at most $k$ literals, this reduces the trivial upper bound of 2 to the sharp bound of $2(1 - 2^{-k})$.

Looking at these results on dependent randomized rounding, we feel that the option of designing the random experiment in such a way that it reflects the structure of the underlying problem has not been exploited sufficiently. In this paper we try to move a step forward in this direction. We impose dependencies that are not necessary in the sense of feasibility but that are helpful in minimizing our objective function $d(A, B)$. For the rounding problem studied in the present paper, this improves the bound of $0.82944|\mathcal{R}|$ obtained by independent randomized rounding to $0.5463|\mathcal{R}|$.

In addition to the randomized rounding condition for single variables, we impose dependencies of the type

$$P\left(\sum_{i \in I_k} y_i = \left\lfloor \sum_{i \in I_k} x_i \right\rfloor + 1\right) = \left\{\sum_{i \in I_k} x_i\right\} \tag{2}$$

for some sets $I_k$ (we write $\{r\} := r - \lfloor r \rfloor$ to denote the fractional part of $r$). Hence our roundings are randomized roundings not only concerning the single variables, but also with respect to the sums $\sum_{i \in I_k} x_i$.[2] By choosing suitable dependencies, we obtain the above mentioned result.

We also consider the question of what dependencies can be realized. More formally, we ask how the sets $I_k$ have to be chosen such that for all values of the input variables a randomized rounding satisfying (2) exists. Surprisingly, there is a simple characterization: Such roundings exist if and only if the hypergraph consisting of the sets $I_k$ is totally unimodular. Of course, with this characterization at hand, the search for suitable dependency sets is much easier.

[2] The dependencies (1) used in [23] actually are a special case of this type where it is also required that (in the notation of (2)) $\sum_{i \in I_k} x_i = 1$ hold.

**2. Independent randomized rounding.** For a number $x$ we write $\lfloor x \rfloor$ for the largest integer not exceeding $x$; $\lceil x \rceil$ for the smallest, being not less than $x$; and $\{x\} := x - \lfloor x \rfloor$ for the fractional part of $x$. We say that some random variable $X$ is a *randomized rounding* of $x$ if $\Pr(X = \lfloor x \rfloor + 1) = \{x\}$ and $\Pr(X = \lfloor x \rfloor) = 1 - \{x\}$. In particular, if $x \in [0,1]$, we have $\Pr(X = 1) = x$ and $\Pr(X = 0) = 1 - x$.

We first analyze what can be achieved with independent randomized rounding. We say that $B$ is an *independent randomized rounding* of $A$ if each entry $b_{ij}$ is a randomized rounding of $a_{ij}$ and all these roundings are mutually independent. Let us remark that we do not expect this to be a competitive approach for the digital halftoning problem. In fact, this idea was experimented with already in the 1950s by Goodall [13] (see also Roberts [24]), long before the seminal work of Raghavan and Thompson. At that time, this was known under the name *thresholding with white noise* or *random dither*.

We use the result below both to estimate the effect of adding dependencies to the rounding process and in the proofs of some of our later results. Note that the proof is different from typical randomized rounding applications: Since the boxes are small, using a large deviation bound makes no sense, and one has to compute the expected error exactly.

THEOREM 2.1. *Let $A \in [0,1]^{m\times n}$, $B$ be an independent randomized rounding of $A$, and $B^*$ be an optimal rounding of $A$ (that is, $d(A, B^*)$ is minimal among all roundings of $A$). Then*

$$E(d(A,B)) \le 0.82944|\mathcal{R}|,$$

$$E(d(A,B)) \le 0.75|\mathcal{R}| + d(A,B^*).$$

*Proof.* By linearity of expectation, $E(d(A,B)) = \sum_{R\in\mathcal{R}} E(d(A_R, B_R)) = \sum_{R\in\mathcal{R}} E(|\Sigma A_R - \Sigma B_R|)$. Hence our analysis is reduced to the rounding problem of a single box $R$. The expected rounding error of a box $\binom{a_1\, a_2}{a_3\, a_4}$ is

$$f(a_1,a_2,a_3,a_4) = \sum_{S\subseteq[4]} \prod_{i\in S} a_i \prod_{i\notin S}(1-a_i) \left| |S| - \sum_{i\in[4]} a_i \right|.$$

Let $a_1,\dots,a_4 \in [0,1]$ such that $f(a_1,a_2,a_3,a_4)$ is maximal. Assume $a_1 < a_2$. Let $0 < \varepsilon \le \frac{1}{2}(a_2 - a_1)$. Then

$$f(a_1,a_2,a_3,a_4) < f(a_1+\varepsilon, a_2-\varepsilon, a_3, a_4) \tag{3}$$

is easily computed, contradicting the maximality of $f(a_1,a_2,a_3,a_4)$. Hence $a_1 \ge a_2$. We have $f(a_1,a_2,a_3,a_4) = f(a_{\sigma(1)}, a_{\sigma(2)}, a_{\sigma(3)}, a_{\sigma(4)})$ for all $a_1,\dots,a_4 \in [0,1]$ and all permutations $\sigma \in S_4$. Thus we conclude that $a_i = a_j$ for all $i,j \in [4]$. Finally, we only need to check that $\overline{f} : [0,1] \to \mathbb{R}; a \mapsto f(a,a,a,a)$ never exceeds $0.82944 = \overline{f}(0.4)$. This is not hard to see, as $\overline{f}$ is piecewise a polynomial of degree 5.

For an arbitrary number $x$ denote by $d(x,\mathbb{Z}) := \min\{x - \lfloor x \rfloor, \lceil x \rceil - x\}$ its distance to the integers. Obviously, $\sum_{R\in\mathcal{R}} d(\Sigma A_R, \mathbb{Z})$ is a lower bound for the optimal error $\min_{B^*} d(A,B^*)$. Analyzing $\tilde{f} : [0,1]^4 \to \mathbb{R}$ defined by $\tilde{f}(a_1,a_2,a_3,a_4) = f(a_1,a_2,a_3,a_4) - d(a_1+a_2+a_3+a_4, \mathbb{Z})$ in a similar manner as above yields that $\tilde{f}$ is bounded from above by 0.75. This shows the second bound of Theorem 2.1. ∎

The two bounds of Theorem 2.1 are tight, as shown by matrices with all entries 0.4 and 0.5, respectively.

**3. Nonindependent randomized rounding.** In this section we improve the previous bounds by adding some dependencies to the rounding process. We start with an elementary approach called *joint randomized rounding*, which reduces the chance that neighboring matrix entries are both rounded in the wrong way. This leads to a first improvement in Corollary 3.3. We then add further dependencies leading to our final bound.

**3.1. Joint randomized rounding.**

DEFINITION 3.1 (joint randomized rounding). *Let $a_1, a_2 \in [0,1]$. We say that $(b_1, b_2)$ is a* joint randomized rounding *of $(a_1, a_2)$ if*

(i) *for all $i \in [2]$, $b_i$ is a randomized rounding of $a_i$;*

(ii) *$b_1 + b_2$ is a randomized rounding of $a_1 + a_2$.*

It is clear that joint randomized roundings exist for all $a_1, a_2 \in [0,1]$: Define $(b_1, b_2)$ through

$$\begin{aligned}
\Pr(b_1 = 1 \wedge b_2 = 0) &= a_1,\\
\Pr(b_1 = 0 \wedge b_2 = 1) &= a_2,\\
\Pr(b_1 = 0 \wedge b_2 = 0) &= 1 - a_1 - a_2
\end{aligned}$$

in the case $a_1 + a_2 \le 1$, and

$$\begin{aligned}
\Pr(b_1 = 1 \wedge b_2 = 0) &= 1 - a_2,\\
\Pr(b_1 = 0 \wedge b_2 = 1) &= 1 - a_1,\\
\Pr(b_1 = 1 \wedge b_2 = 1) &= a_1 + a_2 - 1
\end{aligned}$$

in the case $a_1 + a_2 > 1$. The next lemma shows that two joint randomized roundings are superior to four independent randomized roundings in terms of the rounding error.

LEMMA 3.2. *Let $A = (a_{11}, a_{12}, a_{21}, a_{22})$ be a box. Let $(b_{11}, b_{12})$ be a joint randomized rounding of $(a_{11}, a_{12})$, and $(b_{21}, b_{22})$ one of $(a_{21}, a_{22})$ independent of the first. Set $B = (b_{11}, b_{12}, b_{21}, b_{22})$. Let $B^*$ be an optimal rounding of $A$. Then*

$$\begin{aligned}
E(d(A,B)) &\le \tfrac{16}{27} \le 0.5926,\\
E(d(A,B)) &\le 0.5 + d(A, B^*).
\end{aligned}$$

*Proof.* $b_{11} + b_{12}$ is a randomized rounding of $a_{11} + a_{12}$, and the same holds for the second row. Hence all we have to do is to bound the expected deviation of a sum of two independent randomized roundings from the sum of the original values. We show that $g : [0,1]^2 \to \mathbb{R}$ defined by

$$\begin{aligned}
g(x,y) := {} & xy(2 - x - y) + (x(1-y) + (1-x)y)|1 - x - y|\\
& + (1-x)(1-y)(x+y)
\end{aligned}$$

never exceeds 0.5926. As in the proof of Theorem 2.1, we find that $g(x,y)$ is maximal for $x = y$. Maximizing $x \mapsto g(x,x)$ is straightforward and yields maxima at $(\frac{1}{3}, \frac{1}{3})$ and $(\frac{2}{3}, \frac{2}{3})$, both having objective value $\frac{16}{27} \le 0.5926$. The second bound follows from maximizing $\tilde{g}$ defined by $\tilde{g}(x,y) := g(x,y) - d(x+y, \mathbb{Z})$. $\square$

The bounds in Lemma 3.2 are sharp, as shown by matrices having $a_{11} + a_{12} = a_{21} + a_{22} = \frac{1}{3}$ and $a_{11} + a_{12} = a_{21} + a_{22} = \frac{1}{2}$, respectively. Plastering the grid with independent joint randomized roundings already yields a first improvement over independent randomized rounding, as follows.

COROLLARY 3.3. *Let $A \in [0,1]^{m\times n}$. Compute $B \in \{0,1\}^{m\times n}$ by independently obtaining $(b_{i,2j-1}, b_{i,2j})$ from $(a_{i,2j-1}, a_{i,2j})$ by joint randomized rounding for all $i \in [m], j \in [\frac{n}{2}]$, and also independently obtaining $b_{in}$ from $a_{in}$, $i \in [m]$, by usual randomized rounding if $n$ is odd. Then*

$$E(d(A,B)) \leq 0.7111|\mathcal{R}|,$$
$$E(d(A,B)) \leq 0.625|\mathcal{R}| + d(A,B^*),$$

*where $B^*$ shall be an optimal rounding of $A$.*

*Proof.* At least half of the boxes, namely all $R_{ij}$ such that $j$ is odd, are rounded in the manner of Lemma 3.2. The remaining boxes contain four independent randomized roundings. Thus

$$E(d(A,B)) \leq \tfrac{1}{2}\tfrac{16}{27}|\mathcal{R}| + \tfrac{1}{2}0.82944|\mathcal{R}| \leq 0.7111|\mathcal{R}|,$$
$$E(d(A,B)) - d(A,B^*) \leq \tfrac{1}{2}0.5|\mathcal{R}| + \tfrac{1}{2}0.75|\mathcal{R}| \leq 0.625|\mathcal{R}|,$$

by Theorem 2.1 and Lemma 3.2. ☐

**3.2. Block randomized rounding.**

DEFINITION 3.4 (block randomized rounding). *Let $A = (a_{11}, a_{12}, a_{21}, a_{22})$ be a box. We call $B = (b_{11}, b_{12}, b_{21}, b_{22})$ a* block randomized rounding *of $A$ if*

(i) *each single entry of $B$ is a randomized rounding of the corresponding entry of $A$; i.e.,* $\Pr(b_{ij} = 1) = a_{ij}$ *and* $\Pr(b_{ij} = 0) = 1 - a_{ij}$ *for all $i, j \in [2]$;*

(ii) *each pair of neighboring entries has the distribution of the corresponding joint randomized rounding; i.e., in addition to* (i) *we have for all $(i,j),(i',j') \in [2]\times[2]$ such that either $i \neq i'$ or $j \neq j'$,*

$$\Pr(b_{ij} + b_{i'j'} = \lfloor a_{ij} + a_{i'j'}\rfloor + 1) = \{a_{ij} + a_{i'j'}\},$$
$$\Pr(b_{ij} + b_{i'j'} = \lfloor a_{ij} + a_{i'j'}\rfloor) = 1 - \{a_{ij} + a_{i'j'}\};$$

(iii) *the box in total behaves like a randomized rounding; i.e., we have*

$$\Pr(\Sigma B = \lfloor \Sigma A\rfloor + 1) = \{\Sigma A\},$$
$$\Pr(\Sigma B = \lfloor \Sigma A\rfloor) = 1 - \{\Sigma A\}.$$

LEMMA 3.5. *Let $B$ be a block randomized rounding of a $2\times 2$ matrix $A$, and $B^*$ an optimal rounding of $A$. Then*

$$E(d(A,B)) \leq 0.5,$$
$$E(d(A,B)) \leq 0.125 + d(A,B^*).$$

*Proof.* This follows as a direct consequence of item (iii) of the definition. ☐

Both bounds are sharp, as demonstrated by matrices $A$ such that $\{\sum A\} = \frac{1}{2}$ and $\{\sum A\} = \frac{1}{4}$, respectively. The interesting point is that block randomized roundings always exist. In fact, even more complicated roundings exist, as shown by the following definition and lemma.

DEFINITION 3.6 (continuous block randomized rounding). *Let $A \in [0,1]^{m\times n}$. We say that $B$ is a* continuous block randomized rounding *of $A$ if*

(i) *for all $i \in [\frac{m}{2}], j \in [n-1]$, $R := \{2i-1, 2i\} \times \{j, j+1\}$, $B_{|R}$ is a block randomized rounding of $A_{|R}$;*

(ii) *if $m$ is odd, then for all $j \in [n-1]$, $(b_{m,j}, b_{m,j+1})$ is a joint randomized rounding of $(a_{m,j}, a_{m,j+1})$ as in Definition* 3.1*;*

(iii) *$b_{i,j}$ is mutually independent of all $b_{i',j'}$ such that $\lceil i/2\rceil \neq \lceil i'/2\rceil$.*

LEMMA 3.7. *For any $A \in [0,1]^{m\times n}$, a continuous block randomized rounding $B$ of $A$ exists and can be computed in linear time.*

*Proof.* We may assume that $m$ is even. If not, add an extra row to $A$ with all entries zero, compute $B$ as a continuous block randomized rounding of this matrix, and note that $B$ without the last row—which will be all zero—is a continuous block randomized rounding of $A$.

We propose the following randomized algorithm:

```
Input: $A = (a_{ij}) \in [0,1]^{m \times n}$.
Output: $B = (b_{ij}) \in \{0,1\}^{m \times n}$, a continuous block randomized rounding of $A$.
For all odd $i \in [m]$ do {
    Compute $(b_{i,1}, b_{i+1,1})$ as a joint randomized rounding of $(a_{i,1}, a_{i+1,1})$.
    For all $j \in [n-1]$ use the fact that $(b_{i,j}, b_{i+1,j})$ is a joint randomized
        rounding of $(a_{i,j}, a_{i+1,j})$ to compute $(b_{i,j+1}, b_{i+1,j+1})$ such that $B_{R_{ij}}$
        is a block randomized rounding of $A_{R_{ij}}$}.
```

Clearly, the algorithm is correct if the roundings can be computed efficiently in the inner loop. To simplify notation, we assume $i = 1$ and $j = 1$. Let $a_{11}, a_{12}, a_{21}, a_{22} \in [0,1]$. Assume that $(b_{11}, b_{21})$ is a joint randomized rounding of $(a_{11}, a_{21})$. In the remainder of this proof we show that there is a rounding $(b_{12}, b_{22})$ of $(a_{12}, a_{22})$ such that $(b_{11}, b_{12}, b_{21}, b_{22})$ is a block randomized rounding of $(a_{11}, a_{12}, a_{21}, a_{22})$.

Set $s := a_{11} + a_{12} + a_{21} + a_{22}$. We first show that it is enough to consider the case $s \leq 2$. Assume $s > 2$. Let $a'_{ij} := 1 - a_{ij}$ for all $i, j \in [2]$ as well as $b'_{11} = 1 - b_{11}$ and $b'_{21} = 1 - b_{21}$. Then $a'_{11} + a'_{12} + a'_{21} + a'_{22} < 2$ and $(b'_{11}, b'_{21})$ is a joint randomized rounding of $(a'_{11}, a'_{21})$. Let $(b'_{12}, b'_{22})$ be such that $(b'_{11}, b'_{12}, b'_{21}, b'_{22})$ is a block randomized rounding of the corresponding $a'$ values. Set $b_{12} := 1 - b'_{12}$ and $b_{22} := 1 - b'_{22}$. Now $(b_{11}, b_{12}, b_{21}, b_{22})$ is easily shown to be a block randomized rounding of $(a_{11}, a_{12}, a_{21}, a_{22})$.

Hence, from now on let $s \leq 2$. We distinguish a number of cases.

*Case* 1. $s \leq 1$. If $b_{11} = 1$ or $b_{21} = 1$, then set $b_{12} := 0$ and $b_{22} = 0$. Otherwise choose $b_{12}$ and $b_{22}$ with probabilities

$$\begin{aligned}
\Pr(b_{12} = 1 \wedge b_{22} = 0) &= \tfrac{a_{12}}{1 - a_{11} - a_{21}},\\
\Pr(b_{12} = 0 \wedge b_{22} = 1) &= \tfrac{a_{22}}{1 - a_{11} - a_{21}},\\
\Pr(b_{12} = 0 \wedge b_{22} = 0) &= 1 - \tfrac{a_{12} + a_{22}}{1 - a_{11} - a_{21}}.
\end{aligned}$$

We compute that $(b_{11}, b_{12}, b_{21}, b_{22})$ is a block randomized rounding. Note that the "otherwise" case occurs only if $b_{11} + b_{21} = 0$. Since $(b_{11}, b_{21})$ is a joint randomized rounding of $(a_{11}, a_{21})$, this happens with probability $1 - a_{11} - a_{21}$. Hence the probability that $b_{12}$ becomes 1 is this probability of $1 - a_{11} - a_{21}$ times the probability that $b_{12}$ becomes 1 in the "otherwise" case, which is $a_{12}/(1 - a_{11} - a_{21})$. Hence $\Pr(b_{12} = 1) = a_{12}$, as required. Similarly, the remaining probabilities are proven to be correct.

*Case* 2. $1 < s \leq 2$ and $a_{ij} + a_{i'j'} \leq 1$ for all $i, i', j, j' \in [2]$ such that either $i = i'$ or $j = j'$. We compute $b_{12}$ and $b_{22}$ according to the rule

```
If $b_{11} + b_{21} = 0$
then set $b_{12} := 1$ with probability $\frac{a_{12}}{(a_{12}+a_{22})}$;[3] set $b_{22} := 1 - b_{12}$
else if $b_{11} = 1$
then $b_{12} := 0$; set $b_{22} := 1$ with probability $\frac{a_{22}(s-1)}{(a_{12}+a_{22})a_{11}}$
else if $b_{21} = 1$
then $b_{22} := 0$; set $b_{12} := 1$ with probability $\frac{a_{12}(s-1)}{(a_{12}+a_{22})a_{21}}$.
```

[3]We use the phrase "set $y := 1$ with probability $p$" to describe these actions: Generate a number $r \in [0,1]$ uniformly at random. If $r \leq p$, set $y := 1$; else set $y := 0$.

Using the fact that $(b_{11}, b_{21})$ is a joint randomized rounding of $(a_{11}, a_{21})$, we compute exemplary

$$\begin{aligned}\Pr(b_{12}+b_{22}=1) &= \Pr(b_{11}+b_{21}=0)+\Pr(b_{11}=1)\Pr(b_{22}=1 \mid b_{11}=1)\\ &\quad +\Pr(b_{21}=1)\Pr(b_{22}=1 \mid b_{21}=1)\\ &= (1-a_{11}-a_{21})+a_{11}\cdot\frac{a_{22}(s-1)}{(a_{12}+a_{22})a_{11}}+a_{21}\cdot\frac{a_{12}(s-1)}{(a_{12}+a_{22})a_{21}}\\ &= a_{12}+a_{22}.\end{aligned}$$

*Remaining cases.* Let us call $e=(i,j,i',j')\in[2]^4$ an edge if either $i=i'$ or $j=j'$. We call *e heavy* if $a_{ij}+a_{i'j'}>1$ holds. Exploiting symmetry, we continue this case distinction, treating separately three cases such that there is one heavy edge and two cases such that there are two (intersecting) heavy edges. None of these cases is particularly difficult. In fact, the conditional probabilities arising are uniquely determined; hence working them out is an easy exercise. $\square$

Unfortunately, we could not find a more concise way of computing these roundings. But this is more or less an aesthetic problem: Concerning computing times, a division into many simple cases is rather preferable. Assuming the remaining cases proven, we have given a linear time algorithm for computing continuous block randomized roundings.

We shall first analyze the error of continuous block randomized roundings and then turn to the computational aspects of this approach. We use the following variant of the Chernoff inequality, which can be derived, for example, from Appendix A of Alon and Spencer [1].

LEMMA 3.8. *Let $X_1,\dots,X_r$ be mutually independent random variables such that each two values of a fixed $X_i$ differ by no more than d. Let $X=\sum_{i\in[r]}X_i$. Then*

$$\Pr(X>E(X)+\varepsilon)<\exp\left(\frac{-2\varepsilon^2}{d^2r}\right)$$

*holds for all $\varepsilon\ge 0$.*

THEOREM 3.9. *Let B be a continuous block randomized rounding of $A\in[0,1]^{m\times n}$. Then*

(i) *the expected rounding error satisfies $E(d(A,B))\le 0.5463|\mathcal{R}|$;*
(ii) *if $B^*$ is an optimal rounding, then $E(d(A,B))-d(A,B^*)\le 0.3125|\mathcal{R}|$;*
(iii) *for all $\varepsilon>0$, $\Pr\big(d(A,B)>E(d(A,B))+\varepsilon|\mathcal{R}|\big)<3\exp(-\frac{1}{6}\varepsilon^2(m-3))$.*

*Proof.* At least a half of all boxes (those $R_{ij}$ where $i$ is odd) are block randomized roundings. The remaining boxes contain two independent joint randomized roundings. Thus Lemmas 3.2 and 3.5 yield the bounds (i) and (ii):

$$E(d(A,B))\le\frac{1}{2}\cdot\frac{16}{27}|\mathcal{R}|+\frac{1}{2}\cdot\frac{1}{2}|\mathcal{R}|=\frac{59}{108}|\mathcal{R}|,$$

$$E(d(A,B))-d(A,B^*)\le\frac{1}{2}\cdot\frac{1}{2}|\mathcal{R}|+\frac{1}{2}\cdot\frac{1}{8}|\mathcal{R}|=\frac{5}{16}|\mathcal{R}|.$$

To prove the large deviation bound, let $X_i=\sum_{j=1}^{n-1}d(A_{R_{ij}},B_{R_{ij}})$ for $i\in[m-1]$. Then $X_i$ is a nonnegative random variable bounded by $X_i\le 2(n-1)$. Note that $X_i$ is mutually independent of all $X_{i'}$ such that $|i-i'|\ge 3$. For $k\in[3]$ let $I_k=\{i\in$

$[m-1] \mid i \equiv k \pmod 3\}$ and $Y_k = \sum_{i \in I_k} X_i$. From Lemma 3.8 we conclude

$$\begin{aligned} &\Pr(d(A,B) > E(d(A,B)) + \varepsilon(n-1)(m-1)) \\ &\quad \le \Pr(\exists k \in [3] : Y_k > E(Y_k) + \varepsilon(n-1)|I_k|) \\ &\quad < \sum_{k \in [3]} \exp\left(\frac{-2\varepsilon^2(n-1)^2|I_k|^2}{4(n-1)^2|I_k|}\right) \\ &\quad \le 3\exp\left(-\frac{1}{6}\varepsilon^2(m-3)\right). \qquad \square \end{aligned}$$

In the proof we estimated the rounding error by the worst case rounding errors for the two cases of joint and block randomized rounding. Both worst cases cannot occur simultaneously in all corresponding $2 \times 2$ boxes. In his diploma thesis [25], Schnieder shows that, for the matrix $A$ with all entries $\frac{4-\sqrt{10}}{6} \approx 0.1396$, a continuous block randomized rounding $B$ satisfies $E(d(A,B)) = \frac{40\sqrt{10}-112}{27}|\mathcal{R}| \approx 0.536708|\mathcal{R}|$. In fact, this is asymptotically the worst case, as a tedious analysis reveals.

For the same reason, the relative bound is not sharp. Again the loss is not too large: Let $A$ be such that $a_{ij} = 0.4$ if $i \equiv 1,2 \pmod 4$, and $a_{ij} = 0.6$ if $i \equiv 3,4 \pmod 4$. Then $E(d(A,B)) = 0.3|\mathcal{R}| + d(A,B^*)$, where $B^*$ is an optimal rounding of $A$.

The large deviation bound (iii) displays one disadvantage of nonindependent randomized rounding. Due to the dependencies, the rounding error cannot be written as the sum of $\Omega(|\mathcal{R}|)$ independent random variables. One might think that this problem could be overcome by replacing the Chernoff bound by martingale inequalities. The following example shows that the problem of the errors $d(A_{R_{ij}}, B_{R_{ij}})$ not being independent is "real."

Let $A \in \{0, \frac{1}{2}\}^{4 \times n}$ such that $a_{ij} = 0$ if and only if $i \in \{2,3\}$ and $j$ is even. Let $B$ be a continuous block randomized rounding of $A$. Then $B$ is completely determined by $b_{11}$ and $b_{41}$. In particular, we have $d(A,B) = \frac{2}{3}|\mathcal{R}|$ if $b_{11} = b_{41}$, and $d(A,B) = \frac{1}{3}|\mathcal{R}|$ if $b_{11} \ne b_{41}$. Note that both events occur with probability $\frac{1}{2}$ each.

**3.3. Algorithmic properties of continuous block randomized rounding.** Computing matrix roundings with the randomized approach above has some noteworthy advantages.

First, it is fast. Computing $(b_{i,j+1}, b_{i+1,j+1})$ in Lemma 3.7 takes constant time. Therefore the whole matrix rounding can be done in linear time. The problem of computing time can be further addressed with parallel computing (and this is actually an issue when discussing digital halftoning algorithms). Since the roundings of the $2 \times n$ double-rows are independent, it is no problem to assign them to different processors. From the viewpoint of application to digital halftoning, these points are crucial improvements over the algorithm of [2], which has roughly quadratic time complexity (ignoring a polylogarithmic factor). As stated in [2], this is too slow for high-resolution images.

Our algorithm can be derandomized with the method of conditional probabilities. Since the arising conditional probabilities can be computed efficiently, we do not need pessimistic estimators. The computation of the conditional probabilities can be arranged in such a way that the resulting algorithm has linear time complexity. We refer to Srivastav [31] for a recent survey on derandomization issues.

Another advantage is diversity. Suppose that we do not want to find a good rounding with respect to the $2 \times 2$ boxes, but instead with respect to $3 \times 3$ boxes (or with respect to both). We currently have no hint of whether the error with respect to these sets is a better measure for the visual quality of the resulting digital halftoning, but it seems plausible to try this experimentally. Hence we need an algorithm to compute such roundings.

The algorithm of Asano et al. seems not to work very well for this problem. The roundings computed by their algorithm may have error up to $(1.5-9\varepsilon)|\mathcal{R}|$ with respect to the $3 \times 3$ boxes. This is shown by a matrix $A$ with entries $\frac{1}{6} - \varepsilon$ only, which may be rounded to the all-zero matrix. We are also pessimistic that their approach, in general, can be extended to $3 \times 3$ boxes or other larger structures.

In contrast to these difficulties, nonindependent rounding does very well: We may even use a similar rounding as before: Assuming $m, n$ even for simplicity, compute $B$ from $A$ by letting $B_{R_{ij}}$ be a block randomized rounding of $A_{R_{ij}}$ independently for all odd $i \in [m-1]$ and odd $j \in [n-1]$. We call such an $A$ an independent block randomized rounding. Now each $3 \times 3$ box contains exactly one block randomized rounding, two joint randomized roundings, and one single randomized rounding. Since the four values of the block randomized rounding in total behave like a single randomized rounding, as do the two values of each joint randomized rounding, the expected error of a $3 \times 3$ box is just given by Theorem 2.1. We have the following result.

THEOREM 3.10. *With respect to the family $\mathcal{R}_3$ of $3 \times 3$ boxes, an independent block randomized rounding $B$ of $A$ has expected error $d_{\mathcal{R}_3}(A, B) \leq 0.82944|\mathcal{R}_3|$.*

We presented Theorem 3.10 to demonstrate how easily the ideas of this section can be applied to other rounding problems as well. There are different nonindependent randomized roundings achieving slightly better bounds. The reader is encouraged to try to find some on his own. The following characterization might be helpful for this purpose.

## 4. Proof of the characterization.

In this section we put the results presented previously into a more general framework and prove the characterization proposed in the introduction. As should be clear by now, we are looking for randomized roundings that are also "good" with respect to some sets (i.e., sums) of variables. It is convenient to describe such structures through hypergraphs: A *hypergraph* is a pair $\mathcal{H} = (V, \mathcal{E})$ such that $V$ is a finite set and $\mathcal{E} \subseteq 2^V$. The elements of $V$ are called *vertices*, and those of $\mathcal{E}$ *hyperedges* or *edges* for short. In our setting, $V$ will always be some index set for the variables, and $\mathcal{E}$ contains those sets of variables on which we want our roundings to be "good."

DEFINITION 4.1. *Let $\mathcal{H} = (I, \mathcal{E})$ be a hypergraph. Let $x_i \in \mathbb{R}$ for $i \in I$. Let $y_i, i \in I$, be random variables. For $E \subseteq I$ set $x_E := \sum_{i \in E} x_i$ and $y_E := \sum_{i \in E} y_i$. We call $y_E$ a randomized rounding of $x_E$ if*

$$\begin{aligned} \Pr(y_E = \lfloor x_E \rfloor + 1) &= \{x_E\}, \\ \Pr(y_E = \lfloor x_E \rfloor) &= 1 - \{x_E\} \end{aligned}$$

*holds. We say that $(y_i)_{i \in I}$ is a randomized rounding of $(x_i)_{i \in I}$ with respect to the dependency hypergraph $\mathcal{H}$ (or a randomized $\mathcal{H}$-rounding) if $y_E$ is a randomized rounding of $x_E$ for all $E \in \mathcal{E}$ and $y_i$ is a randomized rounding of $x_i$ for all $i \in I$.*

In this language, continuous block randomized roundings are randomized roundings with respect to the hypergraph $\mathcal{H} = ([m] \times [n], \mathcal{E})$ and

$$\begin{aligned} \mathcal{E} = &\{R_{ij} \mid i \in [m-1] \text{ odd}, j \in [n-1]\} \\ &\cup \{\{(i,j), (i,j+1)\} \mid i \in [m], j \in [n-1]\} \\ &\cup \{\{(i,j), (i+1,j)\} \mid i \in [m-1] \text{ odd}, j \in [n]\}. \end{aligned}$$

A hypergraph is said to be *totally unimodular* if its incidence matrix is totally unimodular. Recall that a matrix is totally unimodular if each square submatrix has determinant $-1$, 0, or 1. There is a rather complicated recursive characterization for totally unimodular matrices and hypergraphs due to Seymour [26, 27]. For our purposes, the following easier, though fundamental, result is sufficient.

THEOREM 4.2 (see Ghouila-Houri [12]). *A hypergraph $\mathcal{H} = (V, \mathcal{E})$ is totally unimodular if and only if for each subset $V_0$ of vertices there is a partition $V_1 \dot\cup V_2 = V_0$ such that any hyperedge $E$ satisfies $\big||E \cap V_1| - |E \cap V_2|\big| \le 1$.*

Let us give some intuition to this result. For $V_0 \subseteq V$, $\mathcal{H}_{|V_0} = (V_0, \{E \cap V_0 \mid E \in \mathcal{E}\})$ is an induced subhypergraph of $\mathcal{H}$. We call a hypergraph $\mathcal{H} = (V, \mathcal{E})$ perfectly balanced if there is a partition $V_1 \dot\cup V_2 = V$ of its vertex set such that $\big||E \cap V_1| - |E \cap V_2|\big| \le 1$ holds for all $E \in \mathcal{E}$, i.e., apart from single vertices of odd cardinality hyperedges, each hyperedge contains the same number of vertices in $V_1$ and $V_2$. In this language, the Ghouila-Houri theorem states that $\mathcal{H}$ is totally unimodular if and only if each induced subhypergraph is perfectly balanced. Both from the definition and (easier) from Ghouila-Houri's characterization, one can deduce that $\mathcal{H}$ is totally unimodular if and only if $(V, \mathcal{E} \cup \{\{v\} \mid v \in V\})$ is totally unimodular. A second result we use is the following well-known theorem of Hoffman and Kruskal.

THEOREM 4.3 (see Hoffman and Kruskal [14]). *Let $A \in \mathbb{R}^{m \times n}$ be a totally unimodular matrix. Let $b, b' \in \mathbb{Z}^m$ and $c, c' \in \mathbb{Z}^n$. Then*

$$\{x \in \mathbb{R}^n \mid b \le Ax \le b', c \le x \le c'\}$$

*is an integral polyhedron.*

Hoffman and Kruskal also showed a converse result: An integral matrix $A$ is totally unimodular if the polyhedron $\{x \in \mathbb{R}^n \mid x \ge 0, Ax \le b\}$ is integral for all $b \in \mathbb{Z}^m$. However, we shall not need that much understanding of integral polyhedra. Basically, it suffices to know the above theorem and the fact that any bounded polyhedron is the convex hull of its extremal points. Thus for any $x \in P$ there are $k \le n+1$, $w^{(1)}, \dots, w^{(k)} \in \operatorname{ex}(P)$, $\lambda_1, \dots, \lambda_k \in [0,1]$ such that $\sum_{\ell \in [k]} \lambda_\ell = 1$ and $x = \sum_{\ell \in [k]} \lambda_\ell w^{(\ell)}$. This is the well-known Carathéodory theorem (see Eckhoff [10] for a survey).

THEOREM 4.4. *Let $\mathcal{H} = (I, \mathcal{E})$ be a hypergraph. The following two properties are equivalent:*

(i) *for all $(x_i)_{i \in I}$ there is a randomized $\mathcal{H}$-rounding $(y_i)_{i \in I}$;*

(ii) *$\mathcal{H}$ is totally unimodular.*

*Proof.* Since the vertex set of $\mathcal{H}$ is not relevant, we may conveniently assume that $I = [n]$. Let $\mathcal{H}$ be totally unimodular and $x_1, \dots, x_n \in \mathbb{R}$. Without loss of generality (cf. the remark following Theorem 4.2), we may assume that $\{j\} \in \mathcal{E}$ for all $j \in [n]$. Let $A$ be an incidence matrix of $\mathcal{H}$; i.e., fix an enumeration $\mathcal{E} = \{E_1, \dots, E_m\}$ of the hyperedges and define $A = (a_{ij})$ by $a_{ij} = 1$ if $j \in E_i$, and $a_{ij} = 0$ otherwise. Set $P = \{w \in \mathbb{R}^n \mid \lfloor Ax \rfloor \le Aw \le \lceil Ax \rceil\}$. By definition, $x \in P$. Since the $n \times n$ identity matrix is a submatrix of $A$, $P$ is bounded.

Thus, $x$ is a convex combination of the extremal point of $P$: There are $w^{(1)}, \dots, w^{(k)} \in \operatorname{ex}(P)$, $\lambda_1, \dots, \lambda_k \in [0,1]$ such that $\sum_{\ell \in [k]} \lambda_\ell = 1$ and $x = \sum_{\ell \in [k]} \lambda_\ell w^{(\ell)}$. By Theorem 4.3, all $w^{(\ell)}$ are integral, and in our case we even have $w_j^{(\ell)} \in \{\lfloor x_j \rfloor, \lceil x_j \rceil\}$. Define the random variables $y_1, \dots, y_n$ by setting $y = w^{(\ell)}$ with probability $\lambda_\ell$.

We compute that $y_1, \dots, y_n$ is a randomized rounding of $x_1, \dots, x_n$ with respect to $\mathcal{H}$: Let $E = E_i \in \mathcal{E}$. If $x_E$ is integral, then $(Aw^{(\ell)})_i = x_E$ for all $\ell \in [k]$ by definition

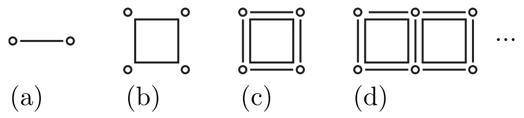


FIG. 1. *Sample diagrams.*

of $P$. Hence $y_E = (Ay)_i$ is trivially a randomized rounding of $x_E$. Assume that $x_E$ is nonintegral. We have $x_E = \sum_{\ell \in [k]} \lambda_\ell (Aw^{(l)})_i$. Set $L_E^+ = \{\ell \in [k] \,|\, (Aw^{(\ell)})_i = \lceil x_E \rceil\}$. Since $(Aw^{(\ell)})_i \in \{\lfloor x_E \rfloor, \lceil x_E \rceil\}$ for all $\ell \in [k]$, we conclude that $\{x_E\} = \sum_{\ell \in L_E^+} \lambda_\ell$. Thus $\Pr(y_E = \lceil x_E \rceil) = \sum_{\ell \in L_E^+} \lambda_\ell = \{x_E\}$. As $y_E$ takes only the two values $\lfloor x_E \rfloor$ and $\lceil x_E \rceil$, we also have $\Pr(y_E = \lfloor x_E \rfloor) = 1 - \{x_E\}$. This shows that $y_1, \ldots, y_n$ is a randomized $\mathcal{H}$-rounding of $x_1, \ldots, x_n$.

Now assume that $\mathcal{H}$ is not totally unimodular. By Theorem 4.2, some induced subhypergraph of $\mathcal{H}$ has discrepancy at least two; i.e., there is a $V_0 \subseteq [n]$ such that for any 2-partition $V_1 \dot\cup V_2 = V_0$ there is an $E \in \mathcal{E}$ such that $\big||E \cap V_1| - |E \cap V_2|\big| \geq 2$. Set $x_j = 0$ for all $j \in [n] \setminus V_0$ and $x_j = \frac{1}{2}$ for all $j \in V_0$. Let $y_1, \ldots, y_n$ be a randomized $\mathcal{H}$-rounding of $x_1, \ldots, x_n$. Then $y_j = 0$ holds with probability one for all $j \in [n] \setminus V_0$ by the definition of randomized $\mathcal{H}$-rounding. For $j \in V_0$, $y_j$ may take only the values 0 and 1. Let $\tilde{y}$ be a possible outcome of the underlying random experiment, that is, an image of the random variable $y$ having probability greater than zero. Let $V_1 = \{j \in V_0 \,|\, \tilde{y}_j = 0\}$ and $V_2 = \{j \in V_0 \,|\, \tilde{y}_j = 1\}$. Let $E \in \mathcal{E}$ such that $\big||E \cap V_1| - |E \cap V_2|\big| \geq 2$. Then $\tilde{y}_E = |E \cap V_2|$, whereas $x_E = \frac{1}{2}|E \cap V_0|$. Thus we have $|x_E - \tilde{y}_E| \geq 1$. Since $|x_E - y_E| \geq 1$ holds with nonzero probability, $y_E$ is not a randomized rounding of $x_E$. This is a contradiction of our assumption that $y_1, \ldots, y_n$ is a randomized $\mathcal{H}$-rounding of $x_1, \ldots, x_n$. $\square$

The use of linear programming ideas to deal with dependencies resembles the work of Koller and Megiddo [18] and Karger and Koller [15]. However, these results are of a completely different nature. There, the aim is to cope with dependencies, since they can lead to smaller sample spaces, whereas we try to use dependencies. There, linear programming ideas are used to transform a given sample space satisfying some dependencies into a smaller one where the dependencies still hold, whereas we construct a sample space directly from a set of given dependencies. Nevertheless, all results indicate that linear programming is a useful framework when dealing with probabilistic constraints.

## 5. Alternative randomized $\mathcal{H}$-roundings.

In this section we investigate what can be achieved by using different randomized $\mathcal{H}$-roundings for the approximation problem of Asano et al. [2]. We omit the proofs for reasons of space, but let us stress that these results depend heavily on Theorem 4.4: It allows us to decide whether a randomized $\mathcal{H}$-rounding exists or not by simply checking whether $\mathcal{H}$ is totally unimodular or not. Thus we may postpone the work of designing a rounding algorithm (and the proof of Lemma 3.7 indicates that this might be quite tedious) until we have found a dependency hypergraph $\mathcal{H}$ yielding suitable bounds.

We denote the hypergraphs considered through simple diagrams. We assume that all variables are randomized roundings. If two variables form a joint randomized rounding, we denote this by connecting the respective nodes by an edge (see Figure 1(a)). If four variables are rounded together (i.e., they form a hyperedge in $\mathcal{H}$), we connect them by a square (Figure 1(b)). Thus a single block randomized rounding is depicted by the diagram in Figure 1(c).

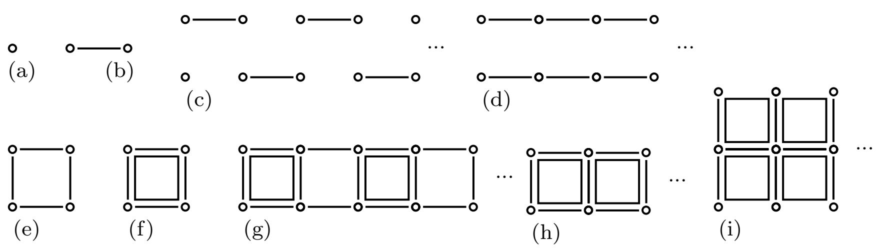


FIG. 2. *Totally unimodular hypergraphs.*

TABLE 1
*Error bounds of the $\mathcal{H}$-roundings depicted in Figure 2.*

| | Absolute | Relative |
|---|---|---|
| (a) | 0.8295 | 0.75 |
| (b) | 0.7111 | 0.625 |
| (c) | 0.75 | 0.5926 |
| (d) | 0.5926 | 0.5 |
| (e) | 0.6518 | 0.5625 |
| (f) | 0.6287 | 0.4688 |
| (g) | 0.5695 | 0.4063 |
| (h) | 0.5493 | 0.3125 |
| (i) | 0.5617 | 0.4166 |

We shall always assume that the grid is plastered with the pattern, and further, that nodes not connected through a series of hyperedges are rounded independently. Hence Figure 1(c) also denotes the independent block randomized rounding investigated in Theorem 3.10. Repeated overlapping patterns will be indicated through (ellipsis) dots. Figure 1(d) thus describes the hypergraph used in section 3.

In Figure 2 we show some totally unimodular hypergraphs. In Table 1 we give upper bounds for the errors inflicted by applying the corresponding $\mathcal{H}$-roundings to Asano's approximation problem. The middle column contains an upper bound for the expected absolute error $E(d(A,B))$, and the right column an upper bound for the expected loss $E(d(A,B)) - d(A,B^*)$ compared to an optimal rounding $B^*$. These upper bounds are derived using the methods of section 3. They are either sharp or close to the optimum.

Table 1 shows several interesting facts. We see that already simple dependencies yield a significant improvement over the independent case (a). Nevertheless, the right choice of dependencies is important: While (b) and (c) look almost identical—plastering the grid with independent joint randomized roundings—the resulting error bounds are not the same. Better bounds can be achieved by larger structures than just independent joint randomized roundings. On the other hand, structures larger than (h) either yield worse bounds, as in (i), or cannot be realized, since the corresponding hypergraph is not totally unimodular. Some examples of the latter type are shown in Figure 3. As Figure 3(c) shows, the dependencies that all boxes are already randomized roundings are not realizable. Thus a nonindependent randomized rounding having $E(d(A,B)) \le \frac{1}{2}|\mathcal{R}|$ seems not to exist. Currently we cannot determine the precise value between our upper bound of $0.5463|\mathcal{R}|$ and the lower one of $\frac{1}{2}|\mathcal{R}|$.

There is some caution necessary in viewing rows (e), (g), and (i). These three patterns generate $\mathcal{R}$-boxes that are not sums of independent randomized roundings. In (e) the error inflicted in the box containing the four nodes connected in a cyclic

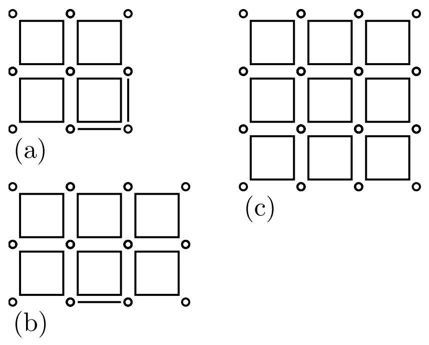


Fig. 3. *Hypergraphs that are not totally unimodular.*

manner depends on the precise way that this rounding is generated. For our estimate we assumed that the nonintersecting joint randomized roundings display neither a positive nor a negative correlation. This is justified by the fact that a rounding exists in which the errors are in fact likely to cancel (namely, (f)). An analogous statement holds for (g) but not for (i). Here Figure 3 tells us that further restrictions on the random process are not possible. Therefore our upper bound here assumes that we have a bad correlation among the horizontally adjacent nodes that do not form a joint randomized rounding.

Note that such negative correlation can indeed occur: Assume three nodes 1, 2, and 3 such that 1 and 2 as well as 2 and 3 form a joint randomized rounding. Let $x_1 = 1 - x_2 = x_3$. Then any nonindependent randomized rounding with respect to these dependencies has $y_1 = y_3$. Thus the error $|(x_1 + x_3) - (y_1 + y_3)|$ does not have the distribution of two independent randomized roundings, but a worse one, namely, twice that of a single randomized rounding.

## 6. Summary and outlook.

This paper describes a new approach in randomized rounding. By imposing suitable dependencies, we improve the expected rounding error significantly. For a problem arising in digital halftoning, this improves previous algorithms with respect to both run-time and rounding error. In particular, we presented the first algorithm that solves this problem fast enough for practical application, namely, in linear time.

On the methodological side, this paper shows that nonindependent randomized rounding can be very effective if one succeeds in finding the right dependencies. To this end, we give a complete characterization of realizable dependencies.

From this work, some open questions arise:

- What is the true approximability of the matrix rounding problem suggested by Asano et al., i.e., can we close the gap between the lower bound of $0.5|\mathcal{R}|$ and the upper bound stemming from our methods?
- For which further rounding problems is it possible to translate part of the structure of the problem into suitable dependencies for the random experiment?
- Engineering aspects: How do the roundings computed by our algorithm look to the human eye? Asano et al. proposed the idea that low error roundings with respect to $2 \times 2$ boxes look nice, but of course they could check this only with their algorithm. Therefore, theoretically, our roundings might even look worse. Coming from the other end, one might (and actually should)

investigate other error measures. Using the ideas of this paper, one may also design rounding algorithms that optimize different error measures stemming from larger boxes or even completely different geometric structures. It could well be that such error measures are better suited to distinguishing good halftonings from poor ones.

**Appendix.** This paper is intended to be of a theoretical nature. Since one of the referees felt that readers from a more applied background would severely miss images and experimental data, we do our best to provide some in this appendix.

We applied the three classical algorithms mentioned in the introduction, together with independent randomized rounding and our algorithm, to several images. All image data used 1 byte per pixel, resulting in an integer value between 0 and 255. We used two types of input data: Real-world images taken with a digital camera (not shown in this paper) and artificial images produced with commercial imaging software. Naturally, the first type is more suitable to estimating how well the algorithm performs in real-world applications, whereas the second is better suited to demonstrating the particular strengths and weaknesses of an algorithm.

The images displayed in Figure 4 are obtained from halftoning an artificially generated image of size $160 \times 160$ pixels with the standard algorithms of error diffusion, ordered dither, dot diffusion, random dither (independent randomized rounding), and our nonindependent randomized rounding algorithm. In all cases, the standard form of the algorithms has been used. It is known that some algorithms produce better outputs if the inputs are subject to some preprocessing. Since such data is mainly available for the algorithms of error diffusion and ordered dither, we believe that an unbiased comparison might be easier without any preprocessings. As indicated in the first paragraph, the intention of this appendix is merely to give the interested reader an idea of how our ideas could work in practice rather than giving evidence that they are competitive with previous algorithms.

Figure 4 shows that the randomized nature of our algorithm has a positive effect on the generation of unwanted structures and grains. Unwanted structures include all kinds of regular patterns like snakes, crosses, or labyrinths that attract objectionable attention. In particular, the error diffusion and ordered dither algorithms tend to produce those.

Grains emerge if in dark (respectively, light) parts of the picture two or more white (respectively, black) pixels touch and thus build a recognizable block. Randomized rounding is very vulnerable to this problem, which is why it is not used in practice for digital halftoning. On the other end, we find error diffusion, which hardly produces any grains. It seems that algorithms that are good concerning graininess tend to produce unwanted structures and vice versa. In this sense, nonindependent randomized rounding could be a fair compromise: Being by far less grainy than independent randomized rounding, it is unlikely to produce unwanted structures.

We also computed the actual errors generated by the algorithms on several input images (on roughly 300,000 pixels). The averages values are given in Table A.1. The error measures considered are the following:

*Total error:* The absolute difference in the average intensity of input and output image $\frac{1}{mn}|\sum_{i,j}(a_{ij} - b_{ij})|$.

$2 \times 2$ *error:* The average error in the $2 \times 2$ boxes, that is, $\frac{1}{|\mathcal{R}|}d(A, B)$, where $d(A, B)$ is the error measure proposed by Asano et al.

$3 \times 3$ *error:* The average error in the $3 \times 3$ boxes.

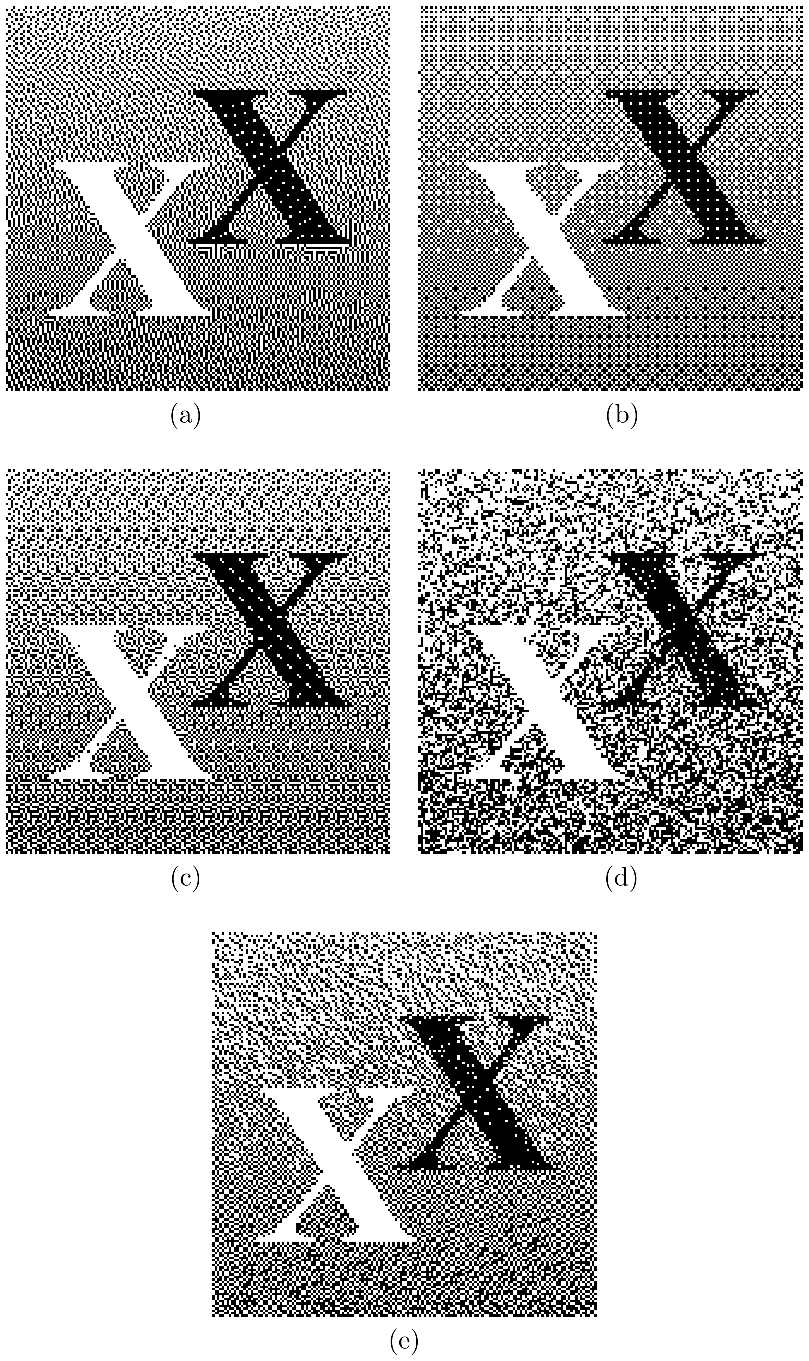


FIG. 4. *Halftoning results of different standard algorithms and our own:* (a) *error diffusion,* (b) *ordered dither,* (c) *dot diffusion,* (d) *random dither (randomized rounding),* (e) *nonindependent randomized rounding.*

Table A.1
*Experimental average errors of the algorithms.*

| Method | Total | $3 \times 3$ | $2 \times 2$ |
|---|---|---|---|
| Error diffusion | 0.00027 | 0.45 | 0.35 |
| Ordered dither | 0.00052 | 0.65 | 0.34 |
| Dot diffusion | 0.00024 | 0.65 | 0.43 |
| Randomized rounding | 0.00056 | 0.96 | 0.64 |
| Nonindependent RR | 0.00034 | 0.60 | 0.40 |

As can be seen, all algorithms change the average intensity only minimally. Concerning the errors in the $2 \times 2$ and $3 \times 3$ boxes, the case is more interesting. Whereas for the $2 \times 2$ boxes error diffusion is slightly, and nonindependent randomized rounding is significantly, worse than the best performer ordered dither, things change for $3 \times 3$ boxes. Here error diffusion takes the lead, followed by nonindependent randomized rounding. Together with the visual quality of the output images, this poses the question of whether larger boxes might yield a better similarity measure. The fact that error diffusion performs relatively well also asks for theoretical bounds for the errors generated with respect to the $2 \times 2$ boxes. Here the inherently sequential nature of this algorithm seems to impose some difficulties.

**Acknowledgments.** I would like to thank Tetsuo Asano, Naoki Katoh, Koji Obokata, and Takeshi Tokuyama for providing me with a preprint of their SODA 2002 paper and some useful discussion on the topic. I am also grateful that Henning Schnieder gave his kind permission for me to use some of his numerical results and sample images in the appendix of this paper.

## REFERENCES

[1] N. Alon and J. H. Spencer, *The Probabilistic Method*, 2nd ed., John Wiley & Sons, New York, 2000.
[2] T. Asano, N. Katoh, K. Obokata, and T. Tokuyama, *Matrix rounding under the $L_p$-discrepancy measure and its application to digital halftoning*, in Proceedings of the 13th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA), San Francisco, 2002, SIAM, Philadelphia, 2002, pp. 896–904.
[3] T. Asano, T. Matsui, and T. Tokuyama, *On the complexities of the optimal rounding problems of sequences and matrices*, in Algorithm Theory (Proceedings of SWAT 2000, Bergen), Lecture Notes in Comput. Sci. 1851, Springer, Berlin, 2000, pp. 476–489.
[4] T. Asano, T. Matsui, and T. Tokuyama, *Optimal roundings of sequences and matrices*, Nordic J. Comput., 7 (2000), pp. 241–256.
[5] B. E. Bayer, *An optimum method for two-level rendition of continuous-tone pictures*, in Conference Record, Proceedings of the IEEE International Conference on Communications, Seattle, WA, IEEE, Piscataway, NJ, 1973, Vol. 1, pp. 11–15.
[6] D. Bertsimas, C. Teo, and R. Vohra, *On dependent randomized rounding algorithms*, Oper. Res. Lett., 24 (1999), pp. 105–114.
[7] B. Doerr, *Linear discrepancy of totally unimodular matrices*, Combinatorica, 24 (2004), pp. 117–125.
[8] B. Doerr, *Non-independent randomized rounding*, in Proceedings of the 14th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA), Baltimore, MD, 2003, SIAM, Philadelphia, 2003, pp. 506–507.
[9] B. Doerr and H. Schnieder, *Non-independent randomized rounding and an application to digital halftoning*, in Proceedings of the European Symposium on Algorithms 2002, Rome, Lecture Notes in Comput. Sci. 2461, R. Möhring and R. Raman, eds., Springer-Verlag, Berlin, Heidelberg, 2002, pp. 399–410.
[10] J. Eckhoff, *Helly, Radon, and Carathéodory type theorems*, in Handbook of Convex Geometry, Vol. A, B, North–Holland, Amsterdam, 1993, pp. 389–448.

[11] R. W. Floyd and L. Steinberg, *An adaptive algorithm for spatial grey scale*, Proceedings of the SID, 17 (1976), pp. 75–77.

[12] A. Ghouila-Houri, *Caractérisation des matrices totalement unimodulaires*, C. R. Acad. Sci. Paris, 254 (1962), pp. 1192–1194.

[13] W. M. Goodall, *Television by pulse code modulation*, Bell Syst. Tech. J., 30 (1951), pp. 33–49.

[14] A. J. Hoffman and J. B. Kruskal, *Integral boundary points of convex polyhedra*, in Linear Inequalities and Related Systems, H. W. Kuhn and A. W. Tucker, eds., Princeton University Press, Princeton, NJ, 1956, pp. 223–246.

[15] D. R. Karger and D. Koller, *(De)randomized construction of small sample spaces in NC*, J. Comput. System Sci., 55 (1997), pp. 402–413.

[16] D. E. Knuth, *Digital halftones by dot diffusion*, ACM Trans. Graphics, 6 (1987), pp. 245–273.

[17] D. E. Knuth, *Two-way rounding*, SIAM J. Discrete Math., 8 (1995), pp. 281–290.

[18] D. Koller and N. Megiddo, *Constructing small sample spaces satisfying given constraints*, SIAM J. Discrete Math., 7 (1994), pp. 260–274.

[19] D. L. Lau and G. R. Arce, *Modern Digital Halftoning*, Signal Processing and Communication, Vol. 8, Marcel Dekker, New York, Basel, 2001.

[20] D. J. Lieberman and J. P. Allebach, *Efficient model based halftoning using direct binary search*, in Proceedings of the 1997 IEEE International Conference on Image Processing, Santa Barbara, CA, IEEE Computer Society, Piscataway, NJ, 1997, pp. 775–778.

[21] B. Lippel and M. Kurland, *The effect of dither on luminance quantization of pictures*, IEEE Trans. Commun. Technol., COM-19 (1971), pp. 879–888.

[22] P. Raghavan, *Probabilistic construction of deterministic algorithms: Approximating packing integer programs*, J. Comput. Syst. Sci., 37 (1988), pp. 130–143.

[23] P. Raghavan and C. D. Thompson, *Randomized rounding: A technique for provably good algorithms and algorithmic proofs*, Combinatorica, 7 (1987), pp. 365–374.

[24] L. G. Roberts, *Picture coding using pseudo-random noise*, IRE Trans. Inform. Theory, IT-8 (1962), pp. 145–154.

[25] H. Schnieder, *Digital Halftoning through Non-independent Randomized Rounding*, Diploma Thesis, Mathematisches Seminar, Christian–Albrechts–Universität zu Kiel, Kiel, Germany, 2003.

[26] P. D. Seymour, *Decomposition of regular matroids*, J. Combin. Theory Ser. B, 28 (1980), pp. 305–359.

[27] P. D. Seymour, *Applications of the regular matroid decomposition*, in Matroid Theory (Szeged, 1982), North-Holland, Amsterdam, 1985, pp. 345–357.

[28] A. Srinivasan, *Approximation algorithms via randomized rounding: A survey*, in Lectures on Approximation and Randomized Algorithms, Advanced Topics in Mathematics (Proceedings of the Berlin-Poznan summer school), M. Karonski, ed., Polish Scientific Publishers, Warsaw, 1999, pp. 9–71.

[29] A. Srinivasan, *Improved approximation guarantees for packing and covering integer programs*, SIAM J. Comput., 29 (1999), pp. 648–670.

[30] A. Srinivasan, *Distributions on level-sets with applications to approximations algorithms*, in Proceedings of the 41th Annual IEEE Symposium on Foundations of Computer Science, Las Vegas, NV, 2001, IEEE, Piscataway, NJ, pp. 588–597.

[31] A. Srivastav, *Derandomization in combinatorial optimization*, in Handbook of Randomized Computing, S. Rajasekaran, P. M. Pardalos, J. H. Reif, and J. D. P. Rolim, eds., Kluwer Academic Publishers, New York, 2001, pp. 731–842.

[32] J. R. Sullivan, L. A. Ray, and R. Miller, *Design of minimum visual modulation halftone patterns*, IEEE Trans. Systems Man. Cybernet., 21 (1991), pp. 33–38.

[33] R. Ulichney, *Digital Halftoning*, MIT Press, Cambridge, MA, 1987.

[34] R. Ulichney, *Dithering with blue noise*, Proc. IEEE, 76 (1988), pp. 56–79.

SIAM J. COMPUT.
Vol. 34, No. 2, pp. 318–332

# INSTABILITY OF FIFO AT ARBITRARILY LOW RATES IN THE ADVERSARIAL QUEUEING MODEL[*]

RAJAT BHATTACHARJEE[†], ASHISH GOEL[‡], AND ZVI LOTKER[§]

**Abstract.** We study the stability of the commonly used packet forwarding protocol, FIFO (first in first out), in the adversarial queueing model. We prove that FIFO can become unstable, i.e., lead to unbounded buffer-occupancies and queueing delays, at arbitrarily low injection rates. In order to demonstrate instability at rate $r$, we use a network of size $\tilde{O}(1/r)$.

**Key words.** FIFO, stability, adversarial queueing model

**AMS subject classifications.** 68M12, 68M20

**DOI.** 10.1137/S0097539703426805

## 1. Introduction.

In traditional queueing theory, the source which generates network traffic and the processing times are typically assumed to be stochastic. However, the growing complexity of network traffic makes it increasingly unrealistic to model traffic as, say, a Poisson stream. Adversarial queueing theory is a robust and elegant framework developed by Borodin et al. [9] to address this problem. In this model, packets are injected into the network by an adversary rather than by a stochastic process. To keep things simple, it is assumed that the route of each packet is given along with the packet itself. Each edge in the network can forward at most one packet in one time step. If there are multiple packets waiting to cross the same edge, then we need a contention resolution protocol to decide which packet goes across and which packets wait in the queue. The adversary is limited in the following way: over any window of $T$ consecutive time steps, the adversary can inject at most $w + rT$ packets that need to traverse any edge in the network. The parameter $r$ is called the injection rate and must be less than 1. The parameter $w$ is called the burst-size. Such an adversary is called a $(w, r)$-adversary. Once injected, packets follow their routes one edge at a time until they reach their destination.

Intuitively, the adversary is not allowed to introduce more traffic on average than $r$ times the capacity of any edge. Thus, there can be no identifiable "hot-spots" in the system. This model finds the fine middle ground between stochastic arrivals on the one hand (as in traditional queueing theory), where packet arrival is too predictable, and completely unconstrained adversaries on the other (as in competitive analysis), where the adversary is allowed to overload the system.

A packet forwarding protocol is said to be *stable* against a given adversary and for a given network if the maximum queue size, as well as the maximum delay experienced

[*]Received by the editors April 28, 2003; accepted for publication (in revised form) April 30, 2004; published electronically December 1, 2004. The instability of FIFO at arbitrarily low rates using polynomial sized networks was first obtained by [8] and the reduction in diameter was obtained by [27]. This paper combines the two results.

http://www.siam.org/journals/sicomp/34-2/42680.html

[†]Department of Computer Science, Stanford University, Stanford CA 94305 (rajatb@stanford.edu). This author's research was supported in part by NSF Award 0133968.

[‡]Department of Management Science and Engineering and (by courtesy) Computer Science, Stanford University, Terman 311, Stanford CA 94305 (ashishg@stanford.edu). This author's research was supported by NSF Career Award 0133968.

[§]Department of Electrical Engineering, Tel Aviv University, Tel Aviv 69978, Israel (zvilo@eng.tau.ac.il).

by a packet, remain bounded. A packet forwarding protocol is said to be stable at rate $r$ (or, $r$-stable) if it is stable against all $(w, r)$-adversaries and for all networks. It is said to be *universally stable* if it is $r$-stable for all $r < 1$. Studying the stability of protocols was the main motivation behind the adversarial queueing theory model. In a seminal paper, Andrews et al. [5] showed that several natural protocols are universally stable, but surprisingly, FIFO (first in first out) is not. This is an important observation since FIFO is by far the most widely used scheduling protocol. It also leaves the following question open:

> Is FIFO $r$-stable at some rate $r > 0$, or is FIFO unstable at arbitrarily low rates?

This is an important question, given the prominence of FIFO as the packet forwarding protocol for the Internet and other networks, and has been the subject of much study over the last few years [12, 10, 11, 5, 4, 23, 25, 28]. For several natural protocols other than FIFO, the corresponding question about the stability threshold in the adversarial model has already been answered [31]. The problem for FIFO is particularly intriguing since there is some intuitive evidence for both the existence and the nonexistence of a stability threshold, due to Bramson [11, 12], arising from two somewhat unrelated models (more details are given in the related work section). Diaz et al. [23] improved the threshold of instability for FIFO to 0.83. (The original proof of Andrews et al. [5] showed instability of FIFO at rate 0.85.) This was further improved by Koukopoulos, Nikoletseas, and Spirakis [25] to 0.749, and recently Lotker, Patt-Shamir, and Rosen [28] further improved it to 0.5.

In this paper, we will assume that packet paths are required to be simple.[1]

**1.1. Our result.** In this paper, we prove that FIFO can become unstable at arbitrarily low rates in the adversarial model. This was one of the major remaining open problems in the field of adversarial queueing theory. In particular, for an arbitrarily low injection rate $r$, we construct a network (with size polynomial in $1/r$) and determine adversarial injections so that FIFO becomes unstable. We then tighten the size of the network to $\tilde{O}(1/r)$. This is quite strong since it even excludes the possibility that FIFO might be stable at rate $O(1/\log^c m)$, where $m$ is the size of the network and $c$ is a constant.[2]

The main idea is the construction of a gadget which, assuming certain initial conditions, allows only a small fraction of packets to pass through it for a long duration. In particular, the fraction of packets which escape is bounded by $k/(1+r)^k$, where $k$ is a parameter of the gadget and can be increased arbitrarily. The network is constructed using this gadget. The adversary works in phases. At the beginning of a phase, we assume that there are some packets waiting to pass through a column of gadgets. Using each gadget in the column, more packets are generated, which want to ultimately traverse a second column. Additional copies of the gadget mentioned above are used to delay and synchronize these new packets so that, at the end of the phase, there are more packets waiting to traverse the second column than were waiting at the first column at the beginning of the phase. Applying this inductively leads to instability.

While the idea of concatenating gadgets to prove instability has been used before (most recently by Lotker, Patt-Shamir, and Rosen [28], for example), our gadget and

[1]Since we will demonstrate the instability of FIFO, any restriction we place on the adversary can only make our results stronger.

[2]It is trivial to see that FIFO (in fact, any greedy protocol) is stable at rate $1/m$.

network are quite different from those used in earlier works [11, 5, 31, 28] to prove instability results. A review of related work is presented below.

**1.2. Related work.** Andrews et al. [5] proved that rings and DAGs (directed acyclic graphs) are universally stable networks (i.e., networks for which all greedy protocols are stable at all rates $r < 1$). They showed that longest in system (LIS) and shortest in system (SIS) are universally stable protocols and that FIFO is not universally stable. In fact, they showed that FIFO can become unstable at rates greater than 0.85. Several natural protocols such as NTG (nearest to go) and LIFO (last in first out) have been shown to be unstable at arbitrarily low rates [31]. Goel [22], Gamarnik [20], and Alvarez, Blesa, and Serna [3] gave a simple and complete characterization of universally stable networks. Diaz et al. [23] improved the threshold of instability for FIFO to 0.83, and Koukopoulos, Nikoletseas, and Spirakis [25] improved it to 0.749. Lotker, Patt-Shamir, and Rosen [28] further improved the instability threshold to all rates above 0.5. Like our construction, they used a network obtained by concatenating parameterized gadgets; of course, their gadget and network are quite different from ours. They also proved that a network with diameter $d$ is stable at all rates below $1/d$.

For the case when routes are not given by the adversary, Aiello et al. [1] and Andrews et al. [6] studied routing algorithms which ensure that no edge gets overloaded, assuming that the adversary injects packets for which this is feasible. Aiello et al. [2] have recently initiated the study of stability in the presence of fixed size buffers. Gamarnik [21] showed that it is undecidable whether a given protocol is universally stable, for an interesting class of protocols. Feige [18] and Dumas [17] demonstrated nonmonotonic phenomena in packet and queueing networks, respectively. Andrews [4] demonstrated the instability of FIFO in session-oriented networks [13, 14].[3]

Stability of networks has also been studied from a more queueing theoretic viewpoint in stochastic networks, where the packets are injected and serviced according to a stochastic process. Instability in stochastic networks was first demonstrated by Rybko and Stolyar [30], building on the work of Lu and Kumar [29] and Kumar and Seidman [26]. The fluid model involves taking the fluid limit of a stochastic process. Dai [15] related stability in the fluid model to that in the stochastic model. Gamarnik [19] proved an analogue of Dai's result for adversarial networks. Dai and Prabhakar [16] studied the stability of a scheduling protocol for data switches with speed-up in the fluid model. Bennett et al. [7] study the bounds on the worst case delay in a network implementing aggregate scheduling, assuming stochastic arrivals.

Bramson studied the stability of FIFO in two different stochastic models. Building on earlier work by Kelly [24] and others, Bramson showed that FIFO is stable at all rates $r < 1$ under a large class of stationary arrival processes, assuming that the time for a packet to traverse an edge is an i.i.d. (independently and identically distributed) exponential random variable [12]. As pointed out in [12], the last assumption is not critical, and the result also holds for generalized Kelly networks. Quite interestingly, Bramson [11] also showed that FIFO can become unstable at arbitrarily low rates in a job-shop scheduling model. Superficially, it would seem that a minor modification of Bramson's techniques could imply our result. However, in Bramson's construction, the same job can visit the same machine multiple times, and have a *different mean processing time* on each visit. In trying to adapt his result to the networking case,

[3]Whether FIFO is unstable at arbitrarily low rates in the session-oriented model remains an interesting open problem.

we run into the problem that different packets queued up at the same link can have different traversal times. Thus the same link appears to be of different length to different packets in a queue. To implement this construction directly requires injecting extra packets to make the link appear "long" to some packets; these extra packets result in a violation of the rate threshold. Hence, a gadget that can delay packets for arbitrarily long durations (like ours) seems inevitable. Even given our gadget, it is not obvious to us how to use a network similar to Bramson's.

Section 2 describes and analyzes our basic gadget. Section 3 gives the construction of a network which is unstable at arbitrarily low injection rates. The adversarial injection patterns are described in section 4. For ease of exposition, we present the proof in two stages. First, we show a family of networks that can be made unstable at all rates $r > 0$. The proof of instability for these networks is simpler to understand, and is presented in section 5. However, in order to achieve instability at rate $r$, the diameter of a network from this family must be $\tilde{O}(1/r^7)$. We then show (section 6) another family of networks, which is more involved but which has a tighter diameter of $\tilde{O}(1/r)$.

**2. The basic gadget.** Section 2.1 describes the topology of the gadget. Section 2.2 talks about a special kind of flow. Finally, in section 2.3 an upper bound on the number of packets that escape the gadget for this flow is proved.

**2.1. The topology of the gadget.** The gadget has a parameter $k$. A $k$-gadget has $2k$ vertices: $v_1, \dots, v_k, w_1, \dots, w_k$. There are four groups of edges:

- input edges, $g_1, \dots, g_k$, pointing into $v_1, \dots, v_k$, respectively;
- output edges, $h_1, \dots, h_k$, pointing out from $w_1, \dots, w_k$, respectively;
- load edges, $e_1, \dots, e_k$, pointing from $v_i$ to $w_i$;
- helper edges, $f_1, \dots, f_k$, pointing from $w_i$ to $v_{i+1}$. The $i$'s wrap around $k$.

Note that the load and helper edges form a ring with the input and output edges pointing in and out from alternate nodes. Figure 1 shows an example gadget.

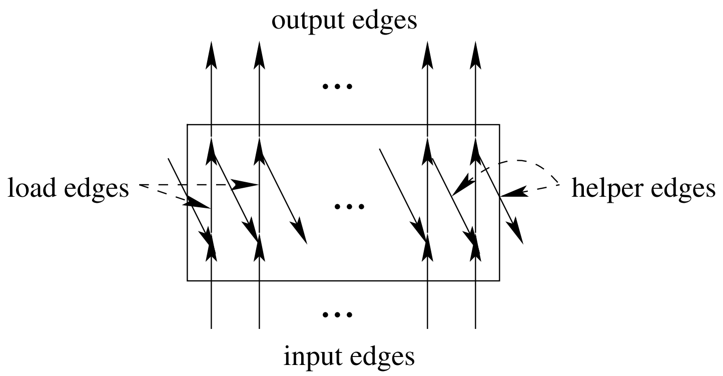


FIG. 1. *A gadget.*

**2.2. A special flow.** There are two kind of packets in this flow:

1. For each $i \in 1, \dots, k$, packets enter the gadget through $g_i$ at rate 1. The route of a particular packet is

$$g_i, e_i, f_i, e_{i+1}, f_{i+1}, e_{i+2}, \dots, f_{i+k-2}, e_{i+k-1}, h_{i+k-1}.$$

Hence, each packet traverses all the load edges in the gadget. These packets are referred to as the *gadget-traversing* packets.

2. For each load edge $e_i$, single edge packets are introduced at a rate $r$. These packets are referred to as the *internal-gadget* packets.

**2.3. Upper bound on the leak.** We are working here in the discrete model. However, we will use the notion of the rate at which packets of a particular kind $X$ arrive at a node, which essentially comes from the fluid model. We can view the rate as the average number of packets arriving at a node per unit time. Also, whenever we use the notion of a rate, we clearly specify the time period on which it is applicable. Rate of leak from a gadget, $R$, is the sum of the rates at which the gadget-traversing packets arrive at the source node of an output edge. In the following, we prove an upper bound for $R$, assuming the flows mentioned in section 2.2. For proving the upper bound we use the following property of FIFO.

*Remark* 2.1. Let $e$ be an edge in the network, and $X_1, X_2, \dots, X_n$ be $n$ different types of packets which arrive at the source node of $e$ at rates $R_1, R_2, \dots, R_n$, respectively. Then the rate at which packets of type $X_i$ traverse $e$ is given by the expression

$$\min\left\{R_i, \frac{R_i}{\sum_{j=1}^{n} R_j}\right\}.$$

LEMMA 2.2. *During the time when the special flow is maintained,* $R \le \frac{k}{(1+r)^k}$.

*Proof.* By symmetry, the total rate of arrival of packets at the source node of the load edges is the same. Let that rate be $T$:

$$T = 1 + r + r_1 + \cdots + r_{k-1},$$

where $r_i$ is the rate of arrival of packets which have traversed $i$ of the $k$ load edges. Using Remark 2.1, we obtain $r_1 = 1/T$. Similarly, for all $2 \le i \le k$,

$$r_i = \frac{r_{i-1}}{T} = \frac{1}{T^i}.$$

Since $T \ge 1 + r$ at all times, it follows that $r_k \le 1/(1+r)^k$. Therefore at all times,

$$R = kr_k \le \frac{k}{(1+r)^k}. \qquad \square$$

Although the above lemma assumes the fluid model, the same bounds can be achieved for the discrete model using slightly higher rates. Also the adversarial injection pattern in this paper is similar to that in the fluid model.

**3. The network topology.** The network contains three components; the columns, the connectors, and the shortcuts. We describe each of these in turn, after first describing the concept of concatenation of gadgets, which is required for each of these components. In this section and all others, all gadgets would have the same parameter, $k$.

**3.1. Concatenating gadgets.** A gadget $G_2$ is said to be concatenated to a gadget $G_1$ if

1. the output edges of $G_1$ act as the load edges of $G_2$, and
2. the load edges of $G_1$ act as the input edges of $G_2$.

Note that more than one gadget can be concatenated to a gadget, and also a single gadget can be concatenated to more than one other gadget. A *chain* $C = \langle H_1, H_2, \ldots, H_n \rangle$, of length $n$, is produced by the concatenation of the gadget $H_{i+1}$ to the gadget $H_i$, for $1 \leq i < n$. A *bridge* $B$ of length $l$ is said to exist between gadgets $G_1$ and $G_2$ if there exists a chain $\langle G_1, H_1, H_2, \ldots, H_l, G_2 \rangle$.

**3.2. Columns.** The network has two separate columns, $C_1, C_2$. A column is a chain of length $\alpha$, where $\alpha$ is a parameter which will be specified later.

$$C_1 = \langle C_{1,1}, C_{1,2}, \ldots, C_{1,\alpha} \rangle,$$

$$C_2 = \langle C_{2,1}, C_{2,2}, \ldots, C_{2,\alpha} \rangle.$$

**3.3. Connectors.** There are two sets of connectors, one from $C_1$ to $C_2$ and the other from $C_2$ to $C_1$. Connectors are bridges of length $\beta$ between each gadget of a column and the first gadget of the other column, where $\beta$ is a parameter which will be specified later. So, for each $C_{1,i}$, we have the following bridge:

$$\langle C_{1,i}, D_{1,i,1}, D_{1,i,2}, \ldots, D_{1,i,\beta}, C_{2,1} \rangle.$$

Similarly, for each $C_{2,i}$, we have the following bridge:

$$\langle C_{2,i}, D_{2,i,1}, D_{2,i,2}, \ldots, D_{2,i,\beta}, C_{1,1} \rangle.$$

**3.4. Shortcuts.** Shortcuts are bridges of length 1 from each connector gadget $D_{1,i,j}$ to $C_{2,1}$. We refer to the respective shortcut gadgets as $E_{1,i,j}$. Similar shortcuts exist between the other set of connector gadgets and $C_{2,1}$. Figure 2 shows the schematic of the network topology.

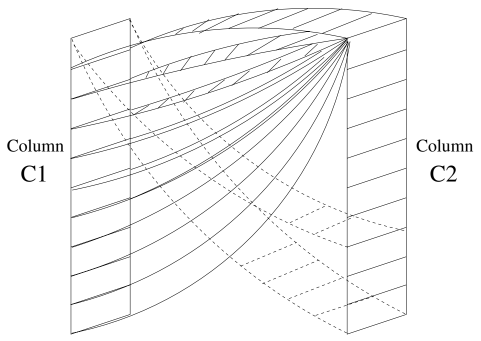


FIG. 2. *Topology of the network; shortcuts are not shown.*

**4. The adversarial injection pattern.** The adversary introduces two kinds of flows of packets (in addition to the internal gadget packets): flow through the columns and flow through the connectors. The description of both assumes the concept of activation of a gadget and sequential activation of gadgets in a chain, which are

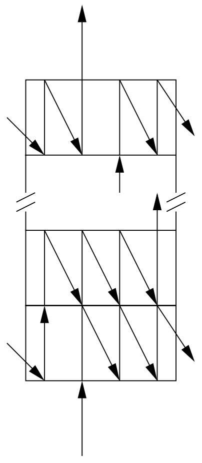


FIG. 3. *Route of a chain-traversing packet.*

described next. We will assume that there is a chain $\langle G_1, G_2, \ldots, G_p \rangle$. These gadgets will be activated one after the other.

A packet is considered to be a *chain-traversing* packet for a chain $\langle G_1, G_2, \ldots, G_p \rangle$ if the route of the packet is gadget-traversing for each $G_i$ in the chain (see Figure 3 for an illustration).

## 4.1. Activation of a gadget.

*Precondition.* There are $t$ gadget-traversing packets in the queue of each input edge of the gadget $G_i$. Also, there is no other packet in any other gadget in the chain. Moreover, the packets are chain-traversing for the chain $\langle G_i, G_{i+1}, \ldots, G_p \rangle$. (We will slightly modify this condition later.)

*Activation.* During the activation phase, the adversary introduces internal-gadget packets at rate $r$ in the gadget $G_i$. The activation phase lasts for $t$ time steps. Note that during this phase the internal-gadget packets and the gadget-traversing packets compete with each other to traverse the load edges of the gadget.

*Postcondition.* There are $t' < t$ packets in the queue of each load edge of the gadget $G_i$, and there are no other packets in any other gadget. Each packet is chain-traversing for the chain $\langle G_{i+1}, G_{i+2}, \ldots, G_p \rangle$. Later a lower bound for the quantity $t'$ will be presented. In order to ensure that these conditions are met, we modify the injection in the following way (we essentially use the fact that the routes of all the packets are predetermined). (1) In order to get rid of internal-gadget packets which might be interleaved with the chain-traversing packets in the queue (after $t$ time steps), we make sure that these particular internal-gadget packets are never introduced. (2) To avoid the case when packets in the queue of the load edges have other edges, within the gadget $G_i$, to traverse, we modify the routes of the packets such that after $t$ time steps, they are simply queued in one of the load edges of $G_i$, and the next edge on their route is a load edge of gadget $G_{i+1}$. This can be done since it amounts to merely taking away some of the edges from the path of the packets and can only reduce the injection rate. (3) We will assume that the routes of all the packets which escaped the gadget during the $t$ time steps end at gadget $G_i$ itself. This ensures that there are no packets in the other gadgets. Note that these changes have no effect on the number of packets queued on a load edge.

*Sequential activation of gadgets.* In a chain of gadgets, the postcondition of the activation of a gadget acts as the precondition for the activation of the concatenated gadget. Sequential activation of gadgets is the cascading activation of gadgets starting from some initial chain-traversing packets waiting in the queue of the input edges of the first gadget.

**4.2. Flow through the columns.** Time steps are grouped into phases. It is assumed that at the beginning of an even phase there are $s$ chain-traversing (for column $C_1$) packets in the queue of each of the input edges of the gadget $C_{1,1}$. We show that at the end of the phase there are more than $s$ chain-traversing (for column $C_2$) packets waiting on each of the input edges of the gadget, $C_{2,1}$. Similarly, at the beginning of an odd phase, there are $s'$ chain-traversing packets in the queue of each of the input edges of the gadget, $C_{2,1}$, and at the end of the phase there are more than $s'$ chain-traversing packets in the queue of each of the input edges of the gadget, $C_{1,1}$. We show packet injections only for the even phases; packet injections for the odd phases are similar. Applying the above repeatedly leads to instability.

*Subphases.* Each phase is divided into $\alpha$ subphases. During the $i$th subphase the gadget $C_{1,i}$ is activated.

**4.3. Flow through the connectors.** At the end of subphase $i$, let there be $s_i$ packets in the queue of each of the input edges of $C_{1,i+1}$. These packets are chain-traversing for column $C_1$. During the next $s_i$ time steps (i.e., the subphase $i+1$), $rs_i/k$ packets each are introduced into the queue of each input edge of $C_{1,i+1}$. These packets are introduced behind the $s_i$ packets mentioned above. The routes of these packets are chosen such that they are chain-traversing for the chain

$$\langle D_{1,i,1}, D_{1,i,2}, \ldots, D_{1,i,j}, E_{1,i,j}, C_{2,1}, C_{2,2}, \ldots, C_{2,\alpha}\rangle$$

for some $j < \beta$. After $s_i$ time steps, these packets form the precondition for the sequential activation of the gadgets in the chain mentioned above. This is because there are $s_i$ packets present in the queue of each input edge of $C_{1,i}$ ahead of the newly injected packets. The routes of the packets are such that at the end of the phase, i.e., at time $\sum_{i=1}^{\alpha} s_i$, they are all queued at the input edges of the the gadget, $C_{2,1}$. This can be achieved using the shortcut gadgets and the fact that the route of each packet is determined in advance ($j < \beta$ can be chosen arbitrarily). Observe that the routes of the chain-traversing packets and the internal-gadget packets introduced during a subphase are mutually exclusive. Moreover, the injection rate of each type of packet is less than $r$ for any edge.

## 5. Proof of instability.

This section shows that at the end of a phase there are more packets queued at the input edges of $C_{2,1}$, than were queued at the input edges of $C_{1,1}$. To initially generate a constant number of packets to start phase 0, we can attach large acyclic graphs to the input edges of $C_{1,1}$, where the acyclic portion is used to generate the initial packets (for details, please refer to [5]).

Analogously to our description of the flow, the analysis can be broken down into two parts: flow through the columns and flow through the connectors.

**5.1. Flow through the columns.** First we set the values of the various parameters. Let $k$ be such that $(1+r)^k > 64k^3/r^2$, $\alpha = 4k/r$, $\beta = 16k^2/r^2$.

LEMMA 5.1. *Let $s_i$ be the duration for which the gadget $C_{1,i}$ remains activated during the ith subphase. For $1 \le i \le \alpha$, the number of packets in the queue of each of the input edges of the gadget $C_{1,i}$ at time $\sum_{j=1}^{i-1} s_j$, i.e., the beginning of the ith subphase, is lower bounded by $s/2$.*

*Proof.* Recall that the duration for which a gadget remains activated is the same as the number of packets waiting at each input edge when the gadget gets activated. Observe that $s_1 = s$. Lemma 2.2 implies that, for a gadget activated for $s_i$ time steps, the total number of packets which leak through in the $s_i$ steps is at most

$$\left\lceil \frac{s_i k}{(1+r)^k} \right\rceil \leq 2s_i \frac{k}{(1+r)^k}.$$

By our choice of $\alpha$ and $k$, it follows that $\alpha\frac{k}{(1+r)^k} < \frac{\alpha r^2}{64k^2} = \frac{r}{16k} < \frac{1}{2}$. We now have

$$\begin{aligned} s_i &\geq s_{i-1}\left(1 - \frac{2k}{(1+r)^k}\right) \\ &\geq s\left(1 - \frac{2k}{(1+r)^k}\right)^{\alpha} \qquad [\text{since } i \leq \alpha] \\ &> s\left(1 - \alpha\frac{2k}{(1+r)^k}\right) \\ &\geq s\left(1 - \frac{1}{2}\right) \\ &= \frac{s}{2}. \end{aligned}$$

Hence, the lemma follows. ◻

**5.2. Flow through the connectors.** Recall that, for each $1 \leq i < \alpha$, a set of $rs_i/k$ (for each edge) chain-traversing packets for the chain

$$\langle C_{1,i}, D_{1,i,1}, D_{1,i,2}, \ldots, C_{2,1}, \ldots, C_{2,\alpha}\rangle$$

is introduced during the subphase $i + 1$.

LEMMA 5.2. *After the packets have activated $j - 1$ gadgets in a connector, the number of packets queued at each input edge of the connector gadget $D_{1,i,j}$ is lower bounded by $rs/4k$.*

*Proof.* Using arguments similar to that used in Lemma 5.1, we can conclude that the number of packets queued at each input edge of the gadget $D_{1,i,j}$ (after $j - 1$ gadgets have been activated) is at least

$$\frac{rs_i}{k}\left(1 - \frac{2k}{(1+r)^k}\right)^{\beta}.$$

By the choice of $\beta$ and $k$, $\beta\frac{2k}{(1+r)^k} < \frac{1}{2}$, and since $s_i > \frac{s}{2}$, the number of packets is more than

$$\frac{rs}{2k}\left(1 - \beta\frac{2k}{(1+r)^k}\right) \geq \frac{rs}{2k}\left(1 - \frac{1}{2}\right) = \frac{rs}{4k}.$$

Hence, the lemma follows. ◻

LEMMA 5.3. *At time $\sum_{i=1}^{\alpha} s_i$, i.e., at the end of all subphases, the number of packets queued at the input edges of $C_{2,1}$ is greater than $s$.*

*Proof.* We first show that $\beta$ is large enough to allow the different sets of chain-traversing packets injected during each of the $\alpha$ subphases to simultaneously arrive at the input edges of $C_{2,1}$. This can be accomplished if the time step at which $D_{1,i,\beta}$

(for all $1 \leq i \leq \alpha$) is activated is greater than $\sum s_i$. By Lemma 5.2, the time step at which $D_{1,i,\beta}$ is activated is more than

$$\frac{rs}{4k}\beta = \frac{rs}{4k}\alpha^2 = \alpha s \geq \sum s_i.$$

Using Lemma 5.2, the number of packets on the queue of each input edge is more than

$$\frac{\alpha rs}{4k} = s.$$

Hence we obtain the lemma. □

We now state the main theorem of our paper.

THEOREM 5.4. *FIFO is unstable for arbitrarily low injection rates.*

*Proof.* In light of Lemma 5.3, we need to show only that the rate of injection is always less than $r$. The latter is true because the gadget-traversing packets and the internal-gadget packets are always injected for mutually exclusive sets of edges and these packets themselves respect the injection rate $r$. (See section 4 for details.) □

**5.3. The size of the network.** We show that the size of the network is polynomial in $\frac{1}{r}$. Let $k$ be $c\frac{1}{r}\log(\frac{1}{r})$ for a large enough $c$. We use the fact that $(1+r)^{\frac{1}{r}} > 2$ when $r < 1$, and obtain.

$$(1+r)^k > 2^{c\log(\frac{1}{r})} \geq \frac{64k^3}{r^2}.$$

Therefore, $k = O(\frac{1}{r}\log(\frac{1}{r}))$. Now, the size of the network is $O(\alpha\beta k) = O(\frac{k^4}{r^3})$, which is polynomial in $\frac{1}{r}$. This is quite strong since it even excludes the possibility that FIFO might be stable at rate $O(\frac{1}{\log^c m})$, where $m$ is the size of the network and $c$ is a constant.

## 6. Reducing the diameter of the network to $\tilde{O}(\frac{1}{r})$.

In this section, we present a variant of our construction in which we prove instability at an arbitrary rate $r > 0$ in a graph of diameter $\tilde{O}(\frac{1}{r})$. (The graph used in the previous section has diameter $\tilde{O}(\frac{1}{r^7})$.) The idea is to combine the gadget in the previous section with results from [28].

**6.1. The topology of the gadget.** We add to our basic $k$-gadget $k$ new edges; we call those edges bypass edges, $b_1, b_2, \ldots, b_k$, pointing from the nodes $v_i$ to the nodes $w_i$, respectively. Note that edges $b_i, e_i$ have the same source (the node $v_i$) and the same target (the node $w_i$). Figure 4 shows two concatenating gadgets.

**6.2. The adversary in two concatenating gadgets.** We use a daisy chain of many gadgets in our construction. However, we start by considering two chained gadgets (Figure 4). We will construct an adversary that maintains the following *gadget invariant*; this condition is similar to Definition 5 in [28]. Let $\alpha = \frac{2r}{2k-r}$.

DEFINITION 6.1. *Let $1 \leq j \leq N$. We say that condition State$(S, \langle H^j, H^{j+1}, \ldots, H^{N+1}\rangle)$ is true at a given time if the following hold at that time on the gadget chain $\langle H^j, H^{j+1}, \ldots, H^{N+1}\rangle$:*

1. *$H^j, H^{j+1}, \ldots, H^{N+1}$ are $k$-gadgets.*
2. *For each $i = 1, \ldots, k$, the number of packets in the buffer of $g_i^j$ is $\alpha \cdot S$, and all those packets in $g_i^j$ have remaining routes $g_i^j, b^j, h_i^j$.*

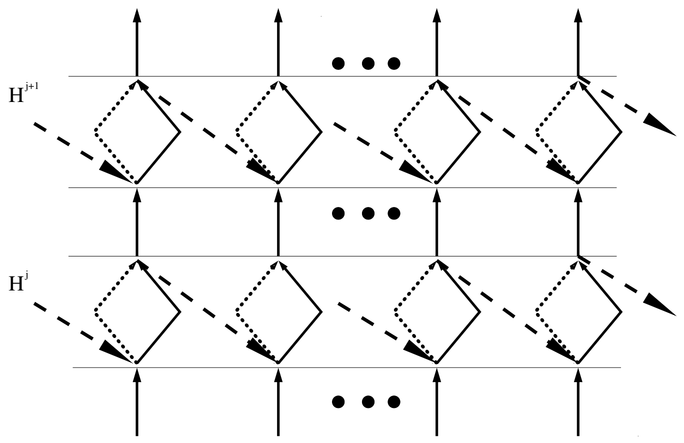


FIG. 4. *A chain of two connected gadgets glued together. The upper gadget is $H^{j+1}$, the lower gadget is $H^j$. Both are $G_k$ gadgets. The twisted dot arrows are the bypass edges $b_i^j, b_i^{j+1}$; the strong twisted arrows are the load edges $e_i^j, e_i^{j+1}$.*

3. *For each $i = 1, \ldots, k$, the number of packets in the buffer of $e_i^j$ is $S$, and all packets in $e_i^j$ have remaining routes $e_i^j, h_i^j$.*
4. *There are no other packets in $H^j, H^{j+1}, \ldots, H^{N+1}$.*
5. *All the packets that are in $H^j$, $H^{j+1}$ can be rerouted in the gadgets $H^{j+2}, H^{j+3}, \ldots, H^{N+1}$ arbitrarily.*

The next lemma shows that if at a time $\tau$ condition $State(S, \langle H^j, H^{j+1}, \ldots, H^{N+1}\rangle)$ holds, then at time $\tau + (1+\alpha)S$ condition $State(S_1, \langle H^{j+1}, \ldots, H^{N+1}\rangle)$ holds, where $\frac{S_1}{S} > 1$. Note that this lemma enables us to use the $k$-gadgets as amplifiers.

LEMMA 6.1. *Let $r > 0$. There exist numbers $k$ and $S_0$ that depend on $r$ such that, for any $S > S_0$, if at some time $\tau$ all the $S$ packets in the gadget $H^j$ were injected after certain time $\tau_0$, and $State(S, \langle H^j, H^{j+1}, \ldots, H^{N+1}\rangle)$ holds at time $\tau$, then there exists a rate $r$ adversary such that, at time $\tau + (1+\alpha)S$, condition $State(S_1, \langle H^{j+1}, \ldots, H^{N+1}\rangle)$ holds for some $S_1 = S(1 + \frac{r}{2k})$.*

*Proof.* For sake of simplicity of presentation we make two facilitating assumptions:

1. $\tau = 0$ and $\tau_0 = \frac{-1}{r}$. This assumption can lead to an additive error of one packet in each load edge of the gadget $H^{j+1}$, and an additional packet from among the packets injected in the load edge of the gadget $H^j$: all together, we get $(k+1)$ fewer packets in the calculation of the number of packets.
2. We ignore floors and ceilings. This is a common assumption in proving instability results (cf. [3, 23, 25]).

We remark that carrying these assumptions throughout the computations would add only additive terms, which can be compensated for by using a larger $S_0$ value (cf. [3, 23, 25, 28]). We now specify the adversary that will create a situation where $State(S_1, \langle H^{j+1}, \ldots, H^{N+1}\rangle)$ holds, as follows.

DEFINITION 6.2. *For each $i = 1, \ldots, k$ the adversary does the following:*

1. *extends the routes of all the packets that enter the gadget $H^{j+1}$ stored in the buffers $g_i^j, e_i^j$ at time 0 by adding the path*

$$e_i^{j+1}, f_i^{j+1}, e_{i+1}^{j+1}, f_{i+1}^{j+1}, e_{i+2}^{j+1}, \ldots, f_{i+k-2}^{j+1}, e_{i+k-1}^{j+1}, h_{i+k-1}^{j+1}$$

*to the path of those packets. Note that the routes of all those packets end in* *the edge $h_i^j$, and therefore we get a legal path. We call these packets* old long *packets.*

2. *injects packets at rate $r$ in time steps $1, 2, \dots, t$. The route of each of these packets is the single edge $e_i^{j+1}$; we call those packets* new short *packets. We set $t$ to be the maximal number such that at time $(1+\alpha)S$ there are no short packets at the edges $e_i^{j+1}$.*
3. (a) *injects packets at rate $\frac{r}{k}$ in the time interval $[1, S]$, with route $e_i^j, h_i^j, b_i^{j+1}, h_i^{j+1}$, for all $i = 1, \dots, k$. We call those packets* new long *packets of the first kind.*
   (b) *injects packets at rate $\frac{r}{k}$ in the time interval $[S+1, (1+\alpha)S]$, with route $h_i^j, b_i^{j+1}, h_i^{j+1}$, for all $i = 1, \dots, k$. We call those packets* new long *packets of the second kind.*

First, note that this is a rate-$r$ adversary: Definition 6.2.1 is justified by Definition 6.1.5. Edges $e_1^{j+1}, \dots, e_k^{j+1}$ are used only by Definition 6.2.2 at rate $r$. Edges $e_i^j, h_i^j, b_i^{j+1}, h_i^{j+1}$ are used only by parts 3(a) and 3(b) of Definition 6.2 at rate $\frac{r}{k}$. Since the time intervals of Definition 6.2 parts 3(a) and 3(b) are disjoint, we get a legal $r$ adversary.

Next we prove that condition $State(S_1, \langle H^{j+1}, \dots, H^{N+1}\rangle)$ holds.

1. Since condition $State(S, \langle H^j, H^{j+1}, \dots, H^{N+1}\rangle)$ holds at time $\tau$, we get that for every $i$, $H^i$ is a $G_k$-gadget.
2. All the new long packets of the first kind leave the edge $e_i^j$ after the last old long packets have left this edge. Since $\frac{r}{k} < \alpha$, all the new long packets of the first kind leave the edge $e_i^j$ by the time $(1+\alpha)S$. Since the number of old long packets is $S \cdot (1+\alpha)$ and all of them arrive at the edge $h_i^j$ before the first new long packet arrives to this edge, no new long packets pass through the edge $h_i^j$ in the time interval $[0, S \cdot (1+\alpha)]$. From Definition 6.2, parts 3(a) and 3(b), the number of new long packets is $S \cdot (1+\alpha) \cdot \frac{r}{k} = S_1 \cdot \alpha$, and all of them are in the edge $h_i^j$. Therefore, Definition 6.1.2 holds.
3. We will see condition 3 of Definition 6.1 in the next subsection.
4. To see that condition 4 of Definition 6.1 holds, we use the idea from [28]. Briefly, this holds because packet routes are predetermined. We can define the destination of long packets that depart from the gadget $H^j$ to be the last node in the gadget $H^j$. To get rid of the short packets at time $(1+\alpha)S$ we simply stop injecting short packets at the right time.
5. To see that Definition 6.1.5 holds, note that the total injection rate of the new long packets is $r$. Therefore we can reroute the path of those packets arbitrarily in the next gadgets. Since the old packets were injected to the graph before time $\frac{-1}{r}$ and the new packets were injected to the graph after time 0, we can reroute the path of all the long packets arbitrarily in the next gadgets. $\square$

**6.3. Upper bound on the leak.** Next we analyze the leak rate from a $k$-gadget $G_k$ in a manner similar to what we did in subsection 2.3. We use a strong version of Lemma 2.2. Let $\phi = \frac{1+\sqrt{5}}{2}$.

LEMMA 6.2. *Assume that we are using the adversary described in Definition* 6.2. *Let $k > 2$; during the time* 0 *to* $(1+\alpha)S$, $R \leq \frac{k}{\phi^k}$.

*Proof.* The proof is similar to the proof of Lemma 2.2 except that, using the fact that $k > 2$ in this case, we can get that $T \geq 1 + r + \frac{1}{T}$ at all times. Solving this inequality, we get that $T > \phi$. From this it follows that $r_k \leq \frac{1}{(\phi)^k}$. Therefore at all

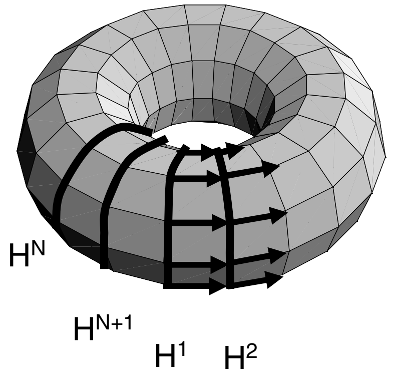


Fig. 5. *The big picture. Every $H^1, H^2, \ldots, H^{N+1}$ is a $G_k$-gadget. In each $G_k$-gadget we have $2k$ nodes that don't appear in this schematic diagram; for accurate description of $G_k$, see subsection 6.1. The marked arrows demonstrate the general direction of packet movement in the networks. The last gadget $H^{N+1}$ serves as a gluing component.*

times, $R = kr_k \leq \frac{k}{\phi^k}$. ◻

The next corollary states that if $k > 4\log_\phi(\frac{1}{r})$ and $\frac{1}{2\sqrt{2}} > r > 0$, we can use the basic gadget $G_k$ as an amplifier.

Lemma 6.3. *Assume that we are using the adversary mentioned in Definition 6.2. Let $\frac{1}{2\sqrt{2}} > r > 0$ and $k > 4\log_\phi(\frac{1}{r})$; then the number of packets in $e_i^{j+1}$ at time $(1+\alpha)S$ is greater than or equal to $S(1+\frac{r}{2k})$.*

*Proof.* Using Lemma 6.2, we get that the total number of packets in $e_i^{j+1}$ at time $(1+\alpha)S$ is $S_1 \geq S \cdot (1+\alpha) \cdot (1-\phi^{-k})$. Using $\frac{1}{2\sqrt{2}} > r > 0$ and $k > 4\log_\phi(\frac{1}{r})$, we get that $\frac{r}{2k} > \frac{1}{\phi^k}$. Now using the definition of $\alpha$, we get that $S_1 \geq S \cdot (\frac{2k+r}{2k-r}) \cdot (\frac{2k-r}{2k}) = S(1+\frac{r}{2k})$. ◻

**6.4. The network topology.** Our networks can be embedded in the torus (see Figure 5). The network contains a chain of gadgets $\langle H^1, \ldots, H^{N+1}\rangle$, and each of these gadgets is a $G_k$-gadget. We use the $H^1, \ldots, H^N$ as amplifiers. We also concatenate the last gadget $H^{N+1}$ to the first gadget $H^1$. We denote this graph by $T_k$. The last gadget acts as gluing component. Now we can repeat exactly what is done in section 3.3 of [28]. Following similar steps, we concatenate a sequence of basic gadgets $H^j$, that is, $1, 2, \ldots, N$, each of which amplifies the number of packets by a multiplicative factor $(1+\frac{r}{2k})$. After this we close the $k$ cycles for each $i = 1, \ldots, k$ exactly as described in Lemma 3.12 of [28]. For completeness we quote Lemma 3.12 from [28], which describes how to close a single cycle. We use this construction for every cycle; i.e., we close $k$ cycles. We consider a path of three edges, called $a_0, a_1$, and $a_2$; we can think of $g_i^{N+1}$ as $a_0$, $e_i^{N+1}$ as $a_1$, and $h_i^{N+1}$ as $a_2$. The routes that will be traversed by old packets will all end at $a_0$, and the fresh packets all start at the tail of $a_2$.

Lemma 6.4. *Suppose that at time $\tau$ there are $S$ packets stored in the buffer of $a_0$ with remaining routes of length 1. Then for any $r > 0$ there exists a rate-$r$ adversary such that at time $\tau + S + rS + r^2S$ there are $r^3S$ packets stored in the buffer of $a_2$ and there are no other packets in the network. Moreover, all the packets stored in the buffer of $a_2$ were injected at the tail of $a_2$ after time $\tau + S$.*

Since the above lemma needs a path of length 3, we can use one gadget $G_k$ in the end. We call this gadget a gluing component. By doing this we lose a fraction of $r^3$ from the total number of packets. The outcome of these two mechanisms is described in Theorem 3.13 of [28]. Finally we get that the total number of packets after traversing all the gadgets $(S_1)$ is $\frac{r}{k(1+\alpha)} \cdot S \cdot (\frac{r}{k})^3 \cdot (1+\frac{r}{2k})^{N-1}$. We show that $S_1 > 10S$. Since $\alpha < 0.5$ and $\frac{1}{k} > r$, we get that $S_1 > \frac{2}{3}S \cdot r^8 \cdot (1+\frac{r}{2k})^{N-1}$. If we take $N = \frac{4k}{r}\log(\frac{4}{r^4}) + 1$, we get that the number of packets in $e_i^1$ at time $t' + S \cdot (\frac{r}{k})^3 \cdot (1+\frac{r}{2k})^{N-1}$ is

$$\begin{aligned} S_1 &> \frac{2}{3} \cdot S \cdot r^8 \cdot \left(1+\frac{r}{2k}\right)^{N-1} \\ &= \frac{2}{3} \cdot S \cdot r^8 \cdot \left(1+\frac{r}{2k}\right)^{\frac{4k}{r}\log\left(\frac{4}{r^4}\right)} \\ &\geq \frac{2}{3} \cdot S \cdot r^8 \cdot 4^{\log\left(\frac{4}{r^4}\right)} \\ &> 10S. \end{aligned}$$

Let $d$ be the diameter of the graph. It is easy to see that $d = O(N \cdot k) = O(\frac{1}{r} \cdot \log^3(\frac{1}{r}))$. In section 4 of Theorem 4.1 in [28] it was proved that FIFO is stable for an injection rate less than $r = \frac{1}{d}$. In this construction we prove that FIFO is unstable for rates bigger than $d = O(\frac{1}{r}\log(\frac{1}{r})^3)$. This leaves a gap of polylog factor.

THEOREM 6.5. *For every $1 > r > 0$ there exists a graph $T_k$, a rate-$r$ adversary, and an initial configuration such that FIFO is unstable on $T_k$ and the diameter of the graph $T_k$ is $O(1/r\log(1/r)^3)$.*

**Acknowledgments.** The third author would like to thank Boaz Patt-Shamir for many helpful discussions.

## REFERENCES

[1] W. Aiello, E. Kushilevitz, R. Ostrovsky, and A. Rosen, *Adaptive packet routing for bursty adversarial traffic*, in Proceedings of the 30th Annual ACM Symposium on Theory of Computing, Dallas, TX, 1998, ACM, New York, pp. 359–368.

[2] W. Aiello, R. Ostrovsky, E. Kushilevitz, and A. Rosén, *Dynamic routing on networks with fixed-size buffers*, in Proceedings of the 14th ACM-SIAM Symposium on Discrete Algorithms, Baltimore, MD, 2003, SIAM, Philadelphia, 2003, pp. 771–780.

[3] C. Alvarez, M. Blesa, and M. Serna, *Universal stability of undirected graphs in the adversarial queueing model*, in Proceedings of the 14th ACM Symposium on Parallel Algorithms and Architectures (SPAA) Winnepeg, MB, 2002, ACM, New York, pp. 183–197.

[4] M. Andrews, *Instability of FIFO in session-oriented networks*, in Proceedings of the 11th Annual ACM-SIAM Symposium on Discrete Algorithms, San Francisco, 2000, SIAM, Philadelphia, 2000, pp. 440–447.

[5] M. Andrews, B. Awerbuch, A. Fernandez, J. Kleinberg, T. Leighton, and Z. Liu, *Universal stability results for greedy contention-resolution protocols*, in Proceedings of the 37th IEEE Symposium on Foundations of Computer Science, Burlington, VT, 1996, IEEE Computer Society, Washington, DC, pp. 380–389.

[6] M. Andrews, A. Fernandez, A. Goel, and L. Zhang, *Source routing and scheduling in packet networks*, in Proceedings of the 42nd IEEE Symposium on Foundations of Computer Science, Las Vegas, NV, 2001, IEEE Computer Society, Washington, DC, pp. 168–177.

[7] J.C.R. Bennett, K. Benson, A. Charny, W.F. Courtney, and J.Y. Le Boudec, *Delay jitter bounds and packet scale rate guarantee expedited forwarding*, IEEE/ACM Trans. Netw., 10 (2002), pp. 529–540.

[8] R. Bhattacharjee and A. Goel, *Instability of FIFO at arbitrarily low rates in the adversarial queueing model*, University of Southern California Technical Report 02-776, available from http://www.cs.usc.edu/tech-reports/technical_reports.html, 2002.

[9] A. Borodin, J.M. Kleinberg, P. Raghavan, M. Sudan, and D.P. Williamson, *Adversarial queueing theory*, J. ACM, 48 (2001), pp. 13–38.

[10] M. Bramson, *Instability of FIFO queueing networks*, Ann. Appl. Probab., 4 (1994), pp. 414–431.

[11] M. Bramson, *Instability of FIFO queueing networks with quick service times*, Ann. Appl. Probab., 4 (1994), pp. 693–718.

[12] M. Bramson, *Convergence to equilibria for fluid models of FIFO queueing networks*, Queueing Systems Theory Appl., 22 (1996), pp. 5–45.

[13] R.L. Cruz, *A calculus for network delay, part* I: *Network elements in isolation*, IEEE Trans. Inform. Theory, 37 (1991), pp. 114–131.

[14] R.L. Cruz, *A calculus for network delay, part* II: *Network analysis*, IEEE Trans. Inform. Theory, 37 (1991), pp. 132–141.

[15] J.G. Dai, *Stability of open multiclass queueing networks via fluid models*, in Stochastic Networks, F. Kelly and R.J. Williams, eds., IMA Vol. Math. Appl. 71, Springer-Verlag, New York, 1995, pp. 71–90.

[16] J.G. Dai and B. Prabhakar, *The throughput of data switches with and without speedup*, in Proceedings of the IEEE INFOCOM, Tel-Aviv, Israel, 2000, IEEE, New York, pp. 556–564.

[17] V. Dumas, *A multiclass network with non-linear, non-convex, non-monotonic stability conditions*, Queueing Systems Theory Appl., 25 (1997), pp. 1–43.

[18] U. Feige, *Nonmonotonic phenomena in packet routing*, in Proceedings of the 31st Annual ACM Symposium on Theory of Computing, Atlanta, GA, 1999, ACM, New York, pp. 583–591.

[19] D. Gamarnik, *Stability of adversarial queues via fluid models*, in Proceedings of the 39th IEEE Symposium on Foundations of Computer Science, Palo Alto, CA, 1998, pp. 60–70.

[20] D. Gamarnik, *Stability of adaptive and nonadaptive packet routing policies in adversarial queueing networks*, SIAM J. Comput., 32 (2003), pp. 371–385.

[21] D. Gamarnik, *On deciding stability of some scheduling policies in queueing systems*, in Proceedings of the 11th Annual ACM-SIAM Symposium on Discrete Algorithms, San Francisco, CA, 2000, SIAM, Philadelphia, 2000, pp. 467–476.

[22] A. Goel, *Stability of networks and protocols in the adversarial queueing model for packet routing*, Networks, 37 (2001), pp. 219–224.

[23] J. Diaz, D. Koukopoulos, S.E. Nikoletseas, M.J Serna, P.G. Spirakis, and D.M. Thilikos, *Stability and non-stability of the FIFO protocol*, in Proceedings of the 13th Annual ACM Symposium on Parallel Algorithms and Architectures, Barcelona, Spain, 2001, ACM, New York, pp. 48–52.

[24] F. Kelly, *Networks of queues with customers of different types*, J. Appl. Probab., 12 (1975), pp. 542–554.

[25] D. Koukopoulos, S.E. Nikoletseas, and P.E. Spirakis, *The Range of Stability for Heterogeneous and FIFO Queueing Networks*, technical report, Electronic Colloquium on Computational Complexity (TR01-099), 2001.

[26] P.R. Kumar and T. Seidman, *Dynamic instabilities and stabilization methods in distributed real-time scheduling of manufacturing systems*, IEEE Trans. Automat. Control, 35 (1990), pp. 289–298.

[27] Z. Lotker, *Note on Instability of FIFO in the Adversarial Queueing Model*, personal communication, 2003.

[28] Z. Lotker, B. Patt-Shamir, and A. Rosén, *New stability results for adversarial queuing*, SIAM J. Comput., 33 (2004), pp. 286–303.

[29] S.H. Lu and P.R. Kumar, *Distributed scheduling based on due dates and buffer policies*, IEEE Trans. Automat. Control, 36 (1991), pp. 1406–1416.

[30] A.N. Rybko and A.L. Stolyar, *Ergodicity of stochastic processes describing the operation of open queueing networks*, Probl. Inf. Transm., 28 (1992), pp. 199–220.

[31] T. Tsaparas, *Stability in Adversarial Queuing Theory*, M.Sc. Thesis, Department of Computer Science, University of Toronto, Toronto, ON, Canada, 1997.

SIAM J. COMPUT. © 2004 Society for Industrial and Applied Mathematics
Vol. 34, No. 2, pp. 333–357

# GENERALIZED IRREDUCIBILITY OF CONSENSUS AND THE EQUIVALENCE OF $t$-RESILIENT AND WAIT-FREE IMPLEMENTATIONS OF CONSENSUS*

TUSHAR CHANDRA†, VASSOS HADZILACOS‡, PRASAD JAYANTI§, AND SAM TOUEG‡

**Abstract.** We study the consensus problem, which requires multiple processes with different input values to agree on one of these values, in the context of asynchronous shared memory systems. Prior research focussed either on $t$-resilient solutions of this problem (which must be correct even if up to $t$ processes crash) or on wait-free solutions (which must be correct despite the crash of any number of processes). In this paper, we show that these two forms of solvability are closely related. Specifically, for all $n > t \geq 2$ and all sets $\mathcal{S}$ of shared object types (that include simple read/write registers), there is a $t$-resilient solution to $n$-process consensus using objects of types in $\mathcal{S}$ if and only if there is a wait-free solution to $(t + 1)$-process consensus using objects of types in $\mathcal{S}$.

Our proof of this equivalence uses another result derived in this paper, which is of independent interest. Roughly speaking, this result states that a wait-free solution to $(n - 1)$-process consensus is never necessary in designing a wait-free solution to $n$-process consensus, regardless of the types of objects available. More precisely, for all $n \geq 2$ and all sets $\mathcal{S}$ of shared object types (that include simple read/write registers), if there is a wait-free solution to $n$-process consensus that uses a wait-free solution to $(n-1)$-process consensus and objects of types in $\mathcal{S}$, then there is a wait-free solution to $n$-process consensus that uses only objects of types in $\mathcal{S}$.

**Key words.** asynchronous distributed computation, consensus, wait-free algorithms, fault tolerant algorithms, impossibility results

**AMS subject classifications.** 68W15, 68Q17

**DOI.** 10.1137/S0097539798344367

## 1. Introduction.

We consider concurrent systems in which asynchronous processes communicate via typed shared objects. Informally, an object's type specifies (i) the number of ports, which represents the maximum number of processes that may access the object *simultaneously*; (ii) the set of states of the object; (iii) the set of operations that processes may apply to the object through its ports; and (iv) the behavior of the object, described by the effect of each operation on the object's state and the value the operation returns, assuming no other operation is accessing the object at that time. Every object is *linearizable* [14]: when operations are invoked concurrently at different ports, the object behaves *as if* each operation had occurred instantaneously, with no interference from other operations, at some point between the time it was invoked and the time it returned its response. An object that belongs to a type with $n$ ports is called *$n$-ported*.

In such systems, some shared objects, such as registers and test&set objects, are

*Received by the editors September 8, 1998; accepted for publication (in revised form) January 13, 2003; published electronically December 1, 2004. A preliminary version of this paper appeared in *Proceedings of the 13th Annual ACM Symposium on Principles of Distributed Computing*, 1994.
http://www.siam.org/journals/sicomp/34-2/34436.html

†IBM T. J. Watson Research Center, Yorktown Heights, NY 10598. Current address: Google, Inc., 1600 Amphitheatre Parkway, Mountain View, CA 94043 (tushar@google.com).

‡Department of Computer Science, University of Toronto, Toronto, ON, MSS 3H5, Canada (vassos@cs.toronto.edu, sam@cs.toronto.edu). The research of the second author was partially supported by a grant from the Natural Sciences and Engineering Research Council of Canada. The research of the fourth author was supported by NSF grant CCR-9102231.

§Department of Computer Science, Dartmouth College, Hanover, NH 03755-3510 (prasad@cs.dartmouth.edu). The research of this author was supported by NSF RIA grant CCR-9410421 and by a Dartmouth College Startup grant.

supported in hardware, while other objects, such as queues and stacks, are implemented in software. Objects used in the implementation of another object (registers and test&set objects, in our example) are called *base* objects with respect to that implementation. We consider two forms of implementations: wait-free and $t$-resilient. An implementation is *wait-free* if every process can complete every operation on the implemented object in a finite number of its own steps, regardless of whether other processes are fast, slow, or have crashed [18, 28, 12]. Wait-free implementations provide an extreme degree of fault-tolerance. They assure that even if just one process survives, it will be able to complete its operations on the implemented object. In contrast, *$t$-resilient* implementations support a more modest degree of fault-tolerance [11, 10, 24]. They guarantee that nonfaulty processes will complete their operations, as long as no more than $t$ processes fail, where $t$ is a specified parameter. It is immediate from the definitions that wait-freedom is equivalent to $(n-1)$-resilience, where $n$ is the number of processes in the system.

This paper concerns $t$-resilient and wait-free implementations of a particular type of object, known as $n$-consensus. Informally, an $n$-consensus object allows each of $n$ processes to access it by proposing a value; the object returns the same value to all accesses, where the value returned is the value proposed by some process. The following are two reasons why it is important to implement $n$-consensus objects:

- It is possible to design a wait-free implementation of an object of *any* type, shared by $n$ processes, using only $n$-consensus objects and registers [12].[1]
- In an asynchronous system, since processes progress at independent and arbitrarily varying speeds, the view that a process holds of the global state of the computation does not necessarily coincide either with the reality or with the views of other processes. However, processes can reconcile their differences and arrive at a mutually acceptable common view if they have access to consensus objects.

A lot of research was aimed at determining the feasibility of implementing $n$-consensus objects from other types of objects. Some of this research studied the feasibility of $t$-resilient implementations [11, 10, 24], while some studied the feasibility of wait-free implementations [24, 12, 9]. To a large degree, the two questions, namely, the feasibility of $t$-resilient implementations and the feasibility of wait-free implementations, were treated separately, as if they had no particular relationship with each other (see section 1.1 for an exception). The main contribution of this paper is to show that the two questions are closely related, as we explain below.

Consider the task of devising a $t$-resilient implementation of an $n$-consensus object. As $n$ decreases, the fraction of nonfaulty processes on which the implementation can rely gets smaller, and the task therefore seems to become progressively more difficult. For example, a $t$-resilient implementation that works only when a majority of processes are nonfaulty cannot be used when $n$ becomes smaller than $2t+1$. In the limit, when $n$ becomes $t+1$, the task amounts to providing a *wait-free* implementation of the object. Thus, *prima faciae*, it seems that the ability of objects belonging to

[1]A register is an object that allows processes to access it (only) through write operations, each of which stores a new value into the register, and read operations, which return the current value of the register. It is possible to implement $k$-ported registers, for any $k$, from just 2-ported registers. This is a consequence of the well-known fact that arbitrary multireader, multiwriter atomic registers (hence, $k$-ported registers, for any $k$) can be implemented from single-reader, single-writer (hence, 2-ported) atomic registers. (For more information on register constructions see, for example, [19, 20, 15].) Because of this, whenever we mention registers we do not bother to explicitly specify the number of ports: it is understood that any number of ports greater than one suffices.

a set $\mathcal{S}$ of types to support a $t$-resilient implementation of an $n$-consensus object is greater than their ability to support a wait-free implementation of a $(t+1)$-consensus object. We show that this is not the case. Specifically, our result is the following.

Equivalence theorem. *For all $n > t \geq 2$ and all sets $\mathcal{S}$ of types that include* `register` *($\mathcal{S}$ may include nondeterministic types), there is a $t$-resilient implementation of an $n$-consensus object from objects of types in $\mathcal{S}$ if and only if there is a wait-free implementation of a $(t+1)$-consensus object from objects of types in $\mathcal{S}$.*

One use of our theorem stems from the observation that proofs of impossibility of $t$-resilient implementations of consensus tend to be generally much harder than proofs of impossibility of wait-free implementations of consensus.[2] Our theorem allows one to conclude the impossibility of $t$-resilient implementations of $n$-consensus simply by establishing the impossibility of wait-free implementations of $(t+1)$-consensus. For instance, since there is no wait-free implementation of 3-consensus from queues and registers [12], our theorem implies that there is no 2-resilient implementation of $n$-consensus from queues and registers (for all $n \geq 3$). We present several other such applications of our theorem.

A key ingredient in our proof of the equivalence theorem discussed above is another result of this paper that is of interest in its own right. Roughly speaking, this result states that $(n-1)$-consensus objects are not necessary for a wait-free implementation of $n$-consensus, *no matter what other base objects may be available for the implementation*. Specifically, our result is as follows.

Generalized irreducibility theorem for consensus. *For all $n \geq 2$ and all sets $\mathcal{S}$ of types that include* `register` *($\mathcal{S}$ may include nondeterministic types), if there is a wait-free implementation of an $n$-consensus object from $(n-1)$-consensus objects and objects of types in $\mathcal{S}$, then there is a wait-free implementation of an $n$-consensus object from objects of types in $\mathcal{S}$.*

*In other words, the $(n-1)$-consensus base objects can be eliminated from the implementation.*

The above theorem is more general (and has a more complex proof) than the well-known irreducibility of consensus, stated as follows: for all $n \geq 2$, there is no wait-free implementation of an $n$-consensus object from $(n-1)$-consensus objects and registers [12, 17]. To illustrate the difference between the two results, we consider the following question: Is there a wait-free implementation of a 4-consensus object from 3-consensus objects, 4-ported queues, and registers? The irreducibility of consensus does not help answer this question, but our result does. Specifically, since there is no wait-free implementation of a 4-consensus object from 4-ported queues and registers [12], our result implies that the answer to the above question is no.

**1.1. Related work.** As stated earlier, this paper has two main results: the generalized irreducibility theorem and the equivalence theorem. We discuss below prior research related to each of these results.

Our generalized irreducibility theorem is related to the following robustness question, posed in [16]: Suppose that (a) there is no wait-free implementation of an $n$-consensus object from registers and objects of type $T$, and (b) there is no wait-free implementation of an $n$-consensus object from registers and objects of type $T'$. Does

[2]The difference in difficulty can be appreciated by comparing the proof that there is no wait-free implementation of a 3-consensus object from registers and 1-bit read-modify-write objects to the proof that there is no 2-resilient implementation of an $n$-consensus object from the same objects [24]. The latter proof is much longer (three pages versus one page) and the arguments are more involved.

it then follow that there is no wait-free implementation of an $n$-consensus object from registers, objects of type $T$, and objects of type $T'$?

For the special case when one of $T$ and $T'$ is $m$-`consensus`, for any $m$, and the other type is arbitrary (it may even be nondeterministic), our generalized irreducibility theorem states that the answer to the robustness question is yes. For the case when both $T$ and $T'$ are deterministic, Borowsky, Gafni, and Afek [5] and Peterson, Bazzi, and Neiger [27] prove that the answer is yes. Neither result is strictly stronger than the other: our result restricts one of the types to be $m$-`consensus`, while theirs restricts both types to be deterministic. It is significant that our result applies to *all* types, not just deterministic ones, because nondeterministic types sometimes exhibit dramatically different properties. For instance, in sharp contrast to the results in [5, 27] stated above, Lo and Hadzilacos prove that the answer to the robustness question is no if types may be nondeterministic [21]. For different models, Chandra et al. [7], Moran and Rappoport [26], and Schenk [30] also prove that the answer to the robustness question is no.

We now describe prior work related to our equivalence theorem. To our knowledge, Borowsky and Gafni are the first to relate $t$-resilient and wait-free implementations of tasks. To state their result, we need to introduce a new object type, called $n$-ported $k$-set consensus. Informally, an object of this type allows each of $n$ processes to access it by proposing a value. Each access returns some value that has been proposed to the object, subject to the requirement that the number of *different* values returned by all the accesses does not exceed $k$. Thus, an $n$-consensus object is the special case of this object where $k = 1$. Borowsky and Gafni's result is that, for all $n > t$, if there is a $t$-resilient implementation of $n$-ported $t$-set consensus from registers, then there is a wait-free implementation of $(t + 1)$-ported $t$-set consensus from registers [3]. It was also shown (independently, in [3, 13, 29]) that there is no wait-free implementation of $(t + 1)$-ported $t$-set consensus from registers. In conjunction with this, Borowsky and Gafni's result implies that, for *all* $n > t$, there is also no $t$-resilient implementation of $n$-ported $t$-set consensus from registers.

Our equivalence theorem differs from Borowsky and Gafni's result in fundamental ways. First, our result concerns $t$-resilient implementations of consensus, while theirs concerns $t$-resilient implementations of $t$-set consensus. Second, our result applies regardless of the types of objects used in the implementation, while their result requires the objects to be registers. The two results are independent in that neither implies the other.

The proofs of both results are based on simulation techniques whereby a small number of processes simulate a $t$-resilient algorithm originally designed for a larger number of processes, in a manner that preserves the resilience of the original algorithm. The two simulation techniques bear some superficial resemblance, but they differ in substance, reflecting the differences between the results noted in the previous paragraph. The simulation we use to obtain the generalized irreducibility theorem (cf. proof of Lemma 5.1) applies only to consensus algorithms, while Borowsky and Gafni's simulation [3, 6] (as well as the simulation we use to obtain the equivalence theorem—cf. proof of Lemma 6.2) applies to a wider class of algorithms that is formally characterized in [6]. Also, the simulations we use to obtain the generalized irreducibility theorem and the equivalence theorem apply to algorithms that use arbitrary base objects, while Borowsky and Gafni's simulation applies only to algorithms that use registers. There is a variant of Borowsky and Gafni's simulation [4, 8] that applies to algorithms that use registers and $k$-set consensus objects. This variant, however, cannot be used to obtain our equivalence theorem because the algorithm that results from the simulation has lower resilience than the original, simulated algorithm.

Our equivalence theorem requires that $t \geq 2$. A result by Lo, strengthened further by Lo and Hadzilacos, proves that the "only if" direction of our theorem does not hold for $t = 1$: There exist nondeterministic [23] and even deterministic [22] object types which, together with registers, can provide a 1-resilient implementation of 3-consensus, but cannot provide a wait-free implementation of 2-consensus.

**1.2. Organization.** In section 2, we describe the model. In sections 3 and 4, we present the intermediate results needed to prove the generalized irreducibility theorem. This theorem and the equivalence theorem are then proved in sections 5 and 6, respectively.

## 2. Model and definitions.

Our description of the model is somewhat informal. Herlihy [12] has shown how to formalize a similar model using I/O automata [25]. We use the following notation for sets of natural numbers: for any $i, j \in \mathbb{N}$, $[i..j] = \{k \in \mathbb{N} : i \leq k \leq j\}$.

**2.1. Types.** A *type* is a tuple $T = (n, OP, RES, Q, \delta)$, where $n$ is a positive integer denoting the number of ports, $OP$ is a set of operations, $RES$ is a set of responses, $Q$ is a set of states, and $\delta \subseteq Q \times OP \times [1..n] \times Q \times RES$ is the type's *sequential specification*. The number of ports corresponds to the maximum number of processes that can concurrently access an object of this type. The sequential specification describes the behavior of an object of this type in the absence of concurrency: If operation $op$ is applied to port $i$ of an object of type $T$ when the object is in state $q$, the object can enter state $q'$ and return response $res$ if and only if $(q, op, i, q', res) \in \delta$. If, for each $(q, op, i) \in Q \times OP \times [1..n]$, the set $\{(q', res) : (q, op, i, q', res) \in \delta\}$ has at most one element, $T$ is deterministic. In this paper we allow types to be nondeterministic, but we require that they exhibit *finite nondeterminism*: each of the above sets must be finite.

For example, the $n$-ported consensus type $n$-`consensus`[3] informally described in section 1 can be formally defined as the tuple $(n, OP, RES, Q, \delta)$, where $OP = \{propose\ u : u \in \mathbb{N}\}$, $RES = \mathbb{N}$, $Q = \mathbb{N} \cup \{\perp\}$, and for each $i \in [1..n]$ and $u, v \in \mathbb{N}$, $\delta$ contains exactly the following tuples: $(\perp, propose\ u, i, u, u)$ and $(v, propose\ u, i, v, v)$. Clearly, this is a deterministic type.

**2.2. Objects and linearizability.** An object $O$ is an instance of a type initialized to a specified state. For each operation $op$ and port $i$ of its type, $O$ provides an access procedure $\text{Apply}(op, i, O)$. This is the sole means by which operation $op$ can be applied to port $i$ of $O$. As explained above, the sequential specification of $O$'s type describes the behavior of $O$ when access procedures are applied sequentially. In general, however, access procedures *at different ports* of an object can be applied concurrently. Usually, the behavior of the object in this case is constrained by the assumption that the object is *linearizable* [14]. This means that if there are no concurrent accesses to the *same* port, then the object behaves *as if* it had been initialized to the specified state and each access procedure occurred instantaneously at some point between its invocation and its response.[4] Occasionally it will be convenient to consider objects

[3] Throughout the paper we use the `typewriter font` for type names.

[4] We emphasize that the linearizability assumption constrains the behavior of an object only if access procedures at the same port are applied sequentially. No assurances are given about what the object does if access procedures are applied concurrently to one of its ports. The object may fail to respond, or it may return arbitrary responses in that case.

that are linearizable only if used in restricted ways. When discussing such objects we will explicitly state the conditions under which they are linearizable.

In general, when we talk about an object we need to specify both its type and its initial state. However, in the case of consensus objects (i.e., objects of type $n$-`consensus` for some $n$) we will assume, without explicitly saying so, that the initial state is $\perp$. This is because a consensus object initialized to any state other than $\perp$ is trivial: it remains in that state forever, returning it to each invocation.

**2.3. Implementation of consensus.** We now explain what an *implementation* of a *target* object $\mathcal{O}$ of type $T$ from a set $A$ of *base* objects is. The concept of implementation can be defined in a general way for target objects of any type. In this paper, however, we are concerned exclusively with implementations of $n$-consensus objects. For simplicity and brevity we tailor our definitions specifically to such objects.

An *implementation of an $n$-consensus object $\mathcal{O}$ from a set $A$ of objects* consists of a specification of access procedures $\text{Apply}(\textit{propose}\ u, i, \mathcal{O})$ for each $u \in \mathbb{N}$ and $i \in [1..n]$. These access procedures can store and manipulate values in private variables using ordinary programming language constructs; in addition they can apply operations (only) to objects in $A$ through the access procedures provided by those objects. To apply operation *op* to port $i$ of a base object $O \in A$, an access procedure of the target object $\mathcal{O}$ invokes $\text{Apply}(op, i, O)$; when this operation finishes it returns a response. A *step* of the target object $\mathcal{O}$'s access procedure refers to the invocation of an operation at some port of a base object, the receipt of that operation's response, and (if relevant) the assignment of that response to a private variable of $\mathcal{O}$'s access procedure. We do not assume a step to be atomic: the invocation of an operation and its response may be separated in time and, during this interval, steps may be performed at other ports of the base object and/or at other base objects. We do, however, assume that a step terminates or, equivalently, that base objects are responsive: once an operation is invoked at a port of a base object, a response is eventually returned.

The implementation must satisfy certain safety properties (typically linearizability) and liveness properties (typically wait-freedom or $t$-resilience). We will state the properties that must be satisfied by (concurrent) executions of $\mathcal{O}$'s access procedures. Before doing so, we need to clarify certain points about such executions.

First, we assume that at most one operation is applied to each port of $\mathcal{O}$. This assumption can be made without loss of generality because $\mathcal{O}$ is a consensus object: If multiple operations could be applied to a port, the response given by a port to the first *propose* operation applied to it can be stored in a private variable associated with the port and returned to any subsequent operation without involving accesses to any other shared objects (recall that operations to the same port must be applied sequentially).

Second, the concurrent executions we consider may contain one or more access procedures of $\mathcal{O}$ that have *not* run to completion. Taking such executions into account is necessary since we are interested in executions where some of the processes that invoke access procedures may crash. Therefore, given an execution, there are four mutually exclusive possibilities for an access procedure $P$ (note that, in view of the previous paragraph, there can be at most one instance of $P$ in an execution):

(a) $P$ does not appear in the execution. This could be because in this execution no process had an interest in invoking $P$, or because the process that has an interest in invoking $P$ has not gotten around to it yet (and perhaps never will because it has crashed).

(b) $P$ is finite and incomplete in the execution, meaning that it has been invoked but has not returned a response. This could be because the process invoking $P$ has not had a chance to run long enough or because it has crashed.

(c) $P$ is complete (and, of course, finite) in the execution, meaning that $P$ has returned a response.

(d) $P$ is infinite (and, of course, incomplete). In this case the process invoking $P$ does not crash (since $P$ has infinitely many steps in the execution), yet the access procedure does not terminate. A typical reason for this behavior is that processes that invoked other procedures crashed at inopportune moments, making it impossible for $P$ to terminate. In this case we will say that the nonterminating access procedure is *blocked* in this execution.

With these comments in mind, consider a concurrent execution $\mathcal{E}$ of a consensus object $\mathcal{O}$'s access procedures. First we will define the safety properties that should hold in $\mathcal{E}$. We say that $\mathcal{O}$

- *satisfies validity in* $\mathcal{E}$ if any value returned by any access procedure in $\mathcal{E}$ was proposed by some access procedure in $\mathcal{E}$;
- *satisfies agreement in* $\mathcal{E}$ if no two values returned by access procedures in $\mathcal{E}$ are different.

Recall that an $n$-consensus object $\mathcal{O}$ is linearizable in $\mathcal{E}$ if the values returned by the access procedures in $\mathcal{E}$ are in accordance with the sequential specification of $n$-`consensus`, when $\mathcal{O}$ is initialized to state $\perp$ and each access procedure in $\mathcal{E}$ takes effect atomically at some point after it is invoked and before it completes. It is easy to see that $\mathcal{O}$ satisfies validity and agreement in $\mathcal{E}$ if and only if it is linearizable in $\mathcal{E}$. Thus, instead of proving that an implementation of $n$-consensus is linearizable, we will prove that it satisfies validity and agreement.

Next we discuss the liveness properties of the implementation. Intuitively, wait-freedom requires that if a process that invokes an access procedure of the target object $\mathcal{O}$ does not crash, then it will receive a response, no matter how many processes invoking access procedures at other ports of $\mathcal{O}$ crash. The property of $t$-resilience requires that if a process that invokes an access procedure of the target object $\mathcal{O}$ does not crash, then it will receive a response, as long as at most $t$ processes invoking access procedures at other ports of $\mathcal{O}$ crash. Actually, there are two slightly different formulations of $t$-resilience. The weaker formulation assumes that all of $\mathcal{O}$'s ports must be accessed in an execution. Thus, if an access procedure does not appear in $\mathcal{E}$, the process that was supposed to invoke it is considered to have crashed. The stronger formulation of $t$-resilience does not make this assumption; here, an access procedure may not appear in $\mathcal{E}$ just because no process had an interest in invoking it in this execution. The difference between these two formulations lies in what counts as one of the up to $t$ crashes that the implementation is supposed to tolerate. With respect to the four aforementioned possibilities (a)–(d) for an access procedure in an execution, in the weaker formulation, access procedures in cases (a) and (b) count as crashes, while in the stronger formulation, only access procedures in case (b) count as crashes. It turns out that our equivalence theorem holds under both definitions of $t$-resilience.

We now state the liveness properties that should hold in $\mathcal{E}$. We say that $\mathcal{O}$

- *is wait-free for port $i$ in* $\mathcal{E}$ if for every $u \in \mathbb{N}$, $\textsc{Apply}(\textit{propose } u, i, \mathcal{O})$ is finite in $\mathcal{E}$;
- *is wait-free in* $\mathcal{E}$ if it is wait-free for each port in $\mathcal{E}$;
- *is weakly $t$-resilient in* $\mathcal{E}$ if the access procedure at some port is infinite only if the access procedures in more than $t$ other ports do not appear or are finite and incomplete in $\mathcal{E}$;

- *is strongly $t$-resilient in $\mathcal{E}$* if the access procedure at some port is infinite only if the access procedures in more than $t$ other ports are finite, nonempty, and incomplete in $\mathcal{E}$.

For each of the safety and liveness properties defined above, we omit the qualifier "in $\mathcal{E}$" if the property holds in every concurrent execution of the target object's access procedures in which the base objects are linearizable.

**2.4. Binding schemes.** The *binding scheme* of an implementation refers to the rules that govern how each access procedure of the implementation's target object can apply operations to ports of the base objects—specifically, the number of ports of a base object to which it can apply operations, and the length of time during which it is permitted to apply operations to these ports. Under the most permissive binding scheme, called *softwired* binding [5], an access procedure can apply operations to any number of ports of a base object, and it "owns" the port only for the duration of each operation. In this binding scheme, different access procedures of the target object may apply (at different times) operations to the same port. Under the more restrictive *one-to-one static binding scheme*, for each access procedure $P$ and each base object $O$ there is at most one port of $O$ to which $P$ can apply operations, in all executions of the implementation; moreover no other access procedure can apply operations to that port of $O$. With one-to-one static binding, we can think of each port of a base object as being "owned" by an access procedure of the target object, namely, the one that is allowed to apply operations to that port.

Unless otherwise specified, in this paper we assume softwired binding.

**2.5. A remark on composing implementations.** An implementation $\mathcal{I}'$ may depend on another implementation $\mathcal{I}$. For instance, suppose that $\mathcal{I}'$ implements an object $\mathcal{O}'$ and this implementation uses, among others, an object $\mathcal{O}$ implemented by $\mathcal{I}$. In this case, an access procedure $P'$ of $\mathcal{O}'$ might include a call to an access procedure $P$ of $\mathcal{O}$. We note that the execution of $P$ should not be viewed as a single step (because a step is required to terminate, but the termination of $P$ may not be necessarily guaranteed by the design of $\mathcal{O}$). The correct view is that $\mathcal{O}$ is implemented from base objects and, hence, the execution of $P$ amounts to performing a sequence of steps on these (responsive) base objects, as dictated by the implementation $\mathcal{I}$. Thus, $\mathcal{O}$ is not viewed as a base object of $\mathcal{O}'$; instead, the base objects of $\mathcal{O}$ are viewed as also belonging to the set of base objects of $\mathcal{O}'$.

## 3. Achieving one-to-one static binding with base consensus objects.

In this section, we prove a result that we will later use in our proof of the generalized irreducibility theorem (section 5). In general, the binding of a target object $\mathcal{O}$ with its base objects is not one-to-one static. The main result of this section is that, if $\mathcal{O}$ has some base consensus objects, then it is possible to transform the implementation so that, in the new implementation, the binding of $\mathcal{O}$ with all its base consensus objects is one-to-one static. We begin by describing an intermediate implementation needed to prove this result.

For any $m > n$, we describe how to implement an $m$-consensus object $\mathcal{O}$ from $n$-consensus objects. The binding of $\mathcal{O}$ with its base objects is one-to-one static, but our implementation is only conditionally correct: It is always wait-free, and it satisfies validity and agreement *provided that* no more than $n$ of the $m$ access procedures of $\mathcal{O}$ take steps.

Consider the $n$-element subsets of $[1..m]$. Let $S_1, S_2, \dots, S_\ell$ be a listing of these subsets, where $\ell = \binom{m}{n}$. For all $i \in S_j$, define $\text{pos}(i, S_j) = k$ if $i$ is the $k$th smallest element in $S_j$.

```
$O_1, O_2, \ldots, O_\ell$: $n$-consensus objects, initialized to $\bot$

$\underline{\textsc{Apply}(\textit{propose } u, i, \mathcal{O})}$;  $u \in \mathbb{N}, i \in [1..m]$

$est_i := u$
for $j := 1$ to $\ell$ do
    if $i \in S_j$ then
        $est_i := \textsc{Apply}(\textit{propose } est_i, \mathrm{pos}(i, S_j), O_j)$
return $est_i$
```

FIG. 1. *Implementation of $m$-consensus object $\mathcal{O}$ from $n$-consensus objects.*

Our implementation of an $m$-consensus object $\mathcal{O}$, described in Figure 1, employs $\ell$ base $n$-consensus objects, denoted $O_1, O_2, \ldots, O_\ell$. The access procedure $\textsc{Apply}(\textit{propose } v, i, \mathcal{O})$ is implemented as follows. For brevity, let $P$ denote this access procedure. $P$ keeps a running estimate of the eventual return value in a local variable $est_i$. Initially, this estimate is $v$, the value that $P$ wants to propose. $P$ considers the base objects $O_1, \ldots, O_\ell$ in sequence and performs the following actions. For each base object $O_j$, $P$ checks whether $i \in S_j$. If $i \notin S_j$, $P$ does not access $O_j$. Otherwise, it proposes its current estimate to $O_j$ (at port $\mathrm{pos}(i, S_j)$ of $O_j$) and regards the return value as its new estimate. After considering all base objects, $P$ regards the estimate as the response of $\mathcal{O}$.

LEMMA 3.1. *The implementation of $m$-consensus object $\mathcal{O}$ in Figure* 1 *has the following properties:*

1. *The binding of $\mathcal{O}$ (with all its base objects) is one-to-one static.*
2. *$\mathcal{O}$ is wait-free.*
3. *$\mathcal{O}$ satisfies validity and agreement in all executions in which at most $n$ access procedures take steps.*

*Proof.* Part 1 follows from the observation that port $i$ of $\mathcal{O}$ applies an operation to port $p$ of base object $O_j$ if and only if $i \in S_j$ and $\mathrm{pos}(i, S_j) = p$. Part 2 follows from the fact that the implementation has no unbounded loops.

For part 3, consider an execution $\mathcal{E}$ where $i_1, i_2, \ldots, i_n$ are all the ports of $\mathcal{O}$ at which access procedures are invoked. (We may assume, without loss of generality, that access procedures in exactly $n$ ports are invoked: if there is an execution that violates validity or agreement and involves access procedures in $n'$ ports, where $n' \leq n$, then there is also an execution that violates validity or agreement, respectively, and involves access procedures in exactly $n$ ports.) Specifically, let $\textsc{Apply}(\textit{propose } u_k, i_k, \mathcal{O})$ be the access procedure executed at port $i_k$; for brevity, let $P_{i_k}$ denote this access procedure. We argue below that $\mathcal{O}$ satisfies validity and agreement in $\mathcal{E}$.

*$\mathcal{O}$ satisfies validity.* Using the fact that the base objects $O_1, \ldots, O_\ell$ satisfy validity, it follows by an easy induction that $\mathcal{O}$ satisfies validity.

*$\mathcal{O}$ satisfies agreement.* Consider the object $O_j$ such that $S_j = \{i_1, i_2, \ldots, i_n\}$. For each $k \in [1..n]$, $P_{i_k}$ accesses $O_j$ in its $j$th iteration of the for-loop. Since $O_j$ returns the same response $u$ to every access procedure that applies a propose operation to it, it follows that all access procedures have the same estimate $u$ at the end of $j$ iterations of the for-loop. That is, for all $k \in [1..n]$, $est_{i_k} = u$ just after $P_{i_k}$ completes the $j$th iteration of the for-loop. Since $O_{j+1}, \ldots, O_\ell$ satisfy validity, it follows from the implementation that the estimate of an access procedure never changes from the $(j+1)$th iteration onward. Thus, for every $k \in [1..n]$, $P_{i_k}$ returns $u$.

This completes the proof of the lemma. □

LEMMA 3.2. *Let $m > n$ and $\mathcal{S}$ be any set of types. Consider a wait-free implementation of an $m$-consensus object $\mathcal{O}$ from objects belonging to types in $\mathcal{S}$. Suppose that $\mathcal{O}$ has an $n$-consensus base object $O$. If the binding of $\mathcal{O}$ with $O$ is not one-to-one static, it is possible to modify the implementation of $\mathcal{O}$ by replacing $O$ with a bounded number of $n$-consensus objects, in such a way that the new implementation satisfies validity and agreement, is wait-free, and the binding of $\mathcal{O}$ with each of the newly introduced $n$-consensus base objects is one-to-one static.*

*Proof.* Suppose that the binding of $\mathcal{O}$ with $O$ is not one-to-one static. Consider the following modifications to the implementation of $\mathcal{O}$:

1. The base objects of the new implementation are the same as in the original implementation with one exception: The base object $O$ is replaced with $O'$, where $O'$ is implemented as in Figure 1. (Thus, $O'$ is an $m$-consensus object, but it is implemented entirely from $n$-consensus objects.)
2. The access procedures of the new implementation are the same as in the original implementation with one exception: Each time an access procedure APPLY(*propose* $v, i, \mathcal{O}$) performs, in the original implementation, an operation (say, *propose* $u$) at some port $j$ of $O$, the new implementation requires the access procedure to perform the same operation (namely, *propose* $u$) at port $i$ of $O'$.

It is obvious from the above modifications that, in the new implementation, port $i$ of $O'$ is used only by the access procedures for port $i$ of $\mathcal{O}$. This, together with the fact that the binding between $O'$ and its base objects is one-to-one static (by part 1 of Lemma 3.1), implies that the binding between $\mathcal{O}$ and the newly introduced $n$-consensus base objects (i.e., the base objects of $O'$) is one-to-one static.

The fact that the new implementation satisfies validity and agreement follows from two facts: (i) $O$, the base object of the old implementation, has only $n$ ports, and (ii) $O'$, which replaces $O$, satisfies validity and agreement if it is accessed at no more than $n$ of its $m$ ports (by part 3 of Lemma 3.1).

That the new implementation is wait-free follows again from two facts: (i) the old implementation is wait-free, and (ii) the implementation of $O'$ is wait-free (by part 2 of Lemma 3.1). This completes the proof of the lemma. □

## 4. The building blocks.

In this section we present three implementations of $n$-consensus objects from $(n-1)$-consensus objects. These implementations are only conditionally correct: each ensures wait-freedom, validity, and agreement only in executions that satisfy certain conditions. Yet they have certain nice properties that make them useful in the proof of the generalized irreducibility theorem, presented in the next section.

### 4.1. Nonconcurrent implementation.

Figure 2 shows an implementation of $n$-consensus object $\mathcal{O}$ from two $(n-1)$-consensus objects $O$ and $O'$ and a register *DEC*. This implementation is wait-free and, if the access procedures for ports $n-1$ and $n$ are not executed concurrently, the implementation satisfies validity and agreement.

The implementation is informally described as follows. For $i \in [1..n]$, let $P_i$ denote the access procedure APPLY(*propose* $u_i, i, \mathcal{O}$), where $u_i \in \mathbb{N}$. $P_1, \ldots, P_{n-2}$ share $\mathcal{O}$'s base objects $O$ and $O'$ with $P_{n-1}$ and $P_n$, respectively. For each $i \in [1..n-2]$, $P_i$ proposes its value $u_i$ to $O$, proposes the return value from $O$ to $O'$, writes the return value from $O'$ in register *DEC* (for "decision"), and returns it. Each of $P_{n-1}$ and $P_n$ first checks whether the return value is already available in *DEC*. If not, it proposes

```
$O, O'$: $(n-1)$-consensus objects, initialized to $\bot$
$DEC$: register, initialized to $\bot$

$\underline{\text{Apply}(propose\ u, i, \mathcal{O})}$; $u \in \mathbb{N}$, $i \in [1..n-2]$

$v_i := \text{Apply}(propose\ u, i, O)$
$DEC := \text{Apply}(propose\ v_i, i, O')$
return $DEC$

$\underline{\text{Apply}(propose\ u, n-1, \mathcal{O})}$; $u \in \mathbb{N}$

if $DEC = \bot$ then
    $DEC := \text{Apply}(propose\ u, n-1, O)$
return $DEC$

$\underline{\text{Apply}(propose\ u, n, \mathcal{O})}$; $u \in \mathbb{N}$

if $DEC = \bot$ then
    $DEC := \text{Apply}(propose\ u, n-1, O')$
return $DEC$
```

FIG. 2. *Nonconcurrent implementation of $n$-consensus object $\mathcal{O}$ from $(n-1)$-consensus objects.*

its value to the appropriate base consensus object (namely, $O$ for $P_{n-1}$ and $O'$ for $P_n$), writes the return value from the base consensus object in *DEC*, and returns it.

We now explain intuitively why this implementation works. Since the implementation needs to be correct only if the steps of $P_{n-1}$ and $P_n$ do not overlap, there are two cases: either $P_{n-1}$ is before $P_n$ or $P_n$ is before $P_{n-1}$. In the former case, $P_1, \ldots, P_{n-2}$ and $P_{n-1}$ agree on a return value using the object $O$, and $P_n$ learns this value simply by reading *DEC*, where the value is made available by $P_{n-1}$. In the latter case, $O'$ serves as the object that brings about agreement on the return value among $P_1, \ldots, P_{n-2}$ and $P_n$, and $P_{n-1}$ learns this value by reading *DEC*.

LEMMA 4.1. *The implementation, shown in Figure 2, of the $n$-consensus object $\mathcal{O}$ from $(n-1)$-consensus objects and a register has the following properties:*

1. *$\mathcal{O}$ is wait-free.*
2. *$\mathcal{O}$ satisfies validity and agreement in all executions in which the access procedures at ports $n-1$ and $n$ of $\mathcal{O}$ are not concurrent.*

*Proof.* Part 1 is obvious. For part 2, let $\mathcal{E}$ be any execution of the implementation; for each $i \in [1..n]$, let $P_i$ be the access procedure at port $i$ of $\mathcal{O}$ that appears in $\mathcal{E}$. (We may assume, without loss of generality, that access procedures at all ports appear in $\mathcal{E}$: If there is any execution that violates validity or agreement, then there is also one that violates validity or agreement, respectively, and involves the access procedures at all ports.)

*$\mathcal{O}$ satisfies validity.* This follows easily by induction and the fact that the base objects $O$ and $O'$ satisfy validity.

*$\mathcal{O}$ satisfies agreement if $P_{n-1}$ and $P_n$ are not concurrent.* Suppose $P_{n-1}$ and $P_n$ are not concurrent in $\mathcal{E}$. We observe that each of $O$ and $O'$ satisfies agreement. Let $d$ and $d'$ denote the (unique) return values from $O$ and $O'$, respectively, in $\mathcal{E}$.

If the first of $P_{n-1}$ and $P_n$ to read *DEC* finds that $DEC \neq \bot$, it is clear from the implementation that some $P_i$, $i \in [1..n-2]$, had previously written $d$ into *DEC*. Since $O$ and $O'$ satisfy agreement, it is easy to see that, in this case, all access procedures return $d$ and so $\mathcal{O}$ satisfies agreement. If the first of $P_{n-1}$ and $P_n$ to read *DEC* finds that $DEC = \bot$, we consider two cases, depending on which of the two access

```
                         $O, O'$: $(n-1)$-consensus objects, initialized to $\bot$
                         $GP$, $SP$, $DEC$: registers, initialized to $\bot$

    APPLY($propose\ u, i, \mathcal{O}$); $u \in \mathbb{N}$,            APPLY($propose\ u, n, \mathcal{O}$); $u \in \mathbb{N}$
                             $i \in [1..n-1]$

1.  $GP :=$ APPLY($propose\ u, i, O$)           $SP := u$
2.  if $SP = \bot$ then                         if $GP = \bot$ then
3.      $vote_i := GP$                              $DEC := u$
4.  else $vote_i := SP$                         else busy-wait until $DEC \neq \bot$
5.  $DEC :=$ APPLY($propose\ vote_i, i, O'$)
6.  return $DEC$                                return $DEC$
```

FIG. 3. *Group-solo implementation of $n$-consensus object $\mathcal{O}$ from $(n-1)$-consensus objects.*

procedures was executed first (recall that $P_{n-1}$ and $P_n$ are not concurrent in $\mathcal{E}$).

If $P_{n-1}$ was executed first, it is clear from the implementation that $P_{n-1}$ writes $d$ (the return value from $O$) in *DEC* and returns it. $P_n$, which begins after $P_{n-1}$ finishes, reads $d$ from *DEC* and returns it. $P_1, \ldots, P_{n-2}$ propose $d$ to $O'$. Since $O'$ satisfies validity, and $d$ is the only value proposed to it, it returns $d$ to all, and all of $P_1, \ldots, P_{n-2}$ therefore return $d$. Thus, $\mathcal{O}$ satisfies agreement.

If $P_n$ was executed first, it is clear from the implementation that $P_1, \ldots, P_{n-2}$ and $P_n$ return $d'$ (the return value from $O'$). $P_n$ also writes $d'$ in *DEC*. $P_{n-1}$, which begins after $P_n$ finishes, reads $d'$ from *DEC*, and returns it. Thus, $\mathcal{O}$ satisfies agreement.

This completes the proof of the lemma. ☐

**4.2. Group-solo implementation.** Figure 3 shows an implementation of an $n$-consensus object $\mathcal{O}$ from two $(n-1)$-consensus objects $O$ and $O'$, and three registers *GP*, *SP*, and *DEC*. This implementation is wait-free for all ports except port $n$. It is also wait-free for port $n$ unless both of the following hold: (i) an operation is invoked on a port other than $n$, and (ii) no such operation completes. (Note that since ports $1, \ldots, n-1$ are wait-free, an operation that has been invoked on these ports can fail to complete only because of a crash.)

We now informally describe how this implementation works. For $i \in [1..n]$, let $P_i$ denote the access procedure at port $i$ of $\mathcal{O}$. $P_1, \ldots, P_{n-1}$ act as one group, while $P_n$ acts as a solo outsider. $P_1, \ldots, P_{n-1}$ reach consensus on their initial proposals by accessing $O$. Each $P_i$ in the group regards the response of $O$ as the group's proposal for consensus with $P_n$ and writes this value into register *GP* (see line 1). (*GP* and *SP* are acronyms for "group's proposal" and "solo proposal," respectively, and *DEC* stands for "decision.") $P_i$ then reads *SP* to check whether the solo access procedure $P_n$ has published its proposal yet. If *SP* is blank, $P_i$ attempts to promote the group's proposal as the consensus value between $P_n$ and the group. Otherwise, $P_i$ attempts to promote the value in *SP* (which is the proposal of the solo access procedure $P_n$) as the consensus value. Lines 2, 3, and 4, in which $P_i$ sets the local variable $vote_i$ to either *GP* or *SP*, implement this strategy. It is possible, however, that some access procedures in the group find *SP* blank and consequently promote the group's proposal, while others find in *SP* a nonblank value that they promote. To reconcile such differences, access procedures in the group reach consensus on their votes by accessing $O'$. The response of $O'$ is regarded as the final outcome of consensus between $P_n$ and the group.

The solo access procedure $P_n$, on the other hand, begins by publishing its proposal in register *SP*. It then reads *GP*, the register where the group's proposal is published. If *GP* is blank, $P_n$ concludes that it is ahead of all access procedures in the group and that access procedures in the group will all vote for its ($P_n$'s) proposal. Thus, $P_n$ regards its proposal as the outcome of its consensus with the group. On the other hand, if $P_n$ finds *GP* nonblank, then $P_n$ is uncertain of the views of the access procedures in the group (because some members of the group might promote $P_n$'s proposal, while the others promote the group's proposal). $P_n$ therefore blocks itself until the consensus value is published in the register *DEC* by (some $P_i$ in) the group.

LEMMA 4.2. *The implementation, shown in Figure* 3*, of the* $n$*-consensus object* $\mathcal{O}$ *from* $(n-1)$*-consensus objects and registers has the following properties:*

1. $\mathcal{O}$ *satisfies validity and agreement.*
2. $\mathcal{O}$ *is wait-free for all ports except port* $n$*.*
3. $\mathcal{O}$ *is wait-free for port* $n$ *in every execution where either no operation is applied to other ports or an operation applied to some port returns a response.*

*Proof.* For part 1, let $\mathcal{E}$ be any execution of the implementation; for each $i \in [1..n]$, let $P_i$ denote the access procedure at port $i$ of $\mathcal{O}$ that appears in $\mathcal{E}$. (As in the proof of Lemma 4.1, we may assume, without loss of generality, that access procedures at all ports appear in $\mathcal{E}$.)

$\mathcal{O}$ *satisfies validity.* This follows easily by induction and the fact that the base objects $O$ and $O'$ satisfy validity.

$\mathcal{O}$ *satisfies agreement.* There are two cases, based on whether the if-condition in line 2 of $P_n$ evaluates to true or to false. Below we consider these cases in turn.

Suppose that the if-condition (in $P_n$'s access procedure) evaluates to true. Then two facts are obvious from the implementation: (i) $P_n$ finished writing its proposal $u$ in *SP* before any of $P_1, \ldots, P_{n-1}$ completed line 1 of its access procedure, and (ii) $P_n$ writes $u$ in *DEC* and returns it. Fact (i) implies that, for each $i \in [1..n-1]$, the if-condition on line 2 of $P_i$ evaluates to false, thus causing $P_i$ to set $vote_i$ to $u$, the value in *SP*. Thus, all of $P_1, \ldots, P_{n-1}$ propose $u$ to $O'$; since $O'$ satisfies validity, $O'$ returns $u$ to all. Thus, all of $P_1, \ldots, P_{n-1}$ return $u$. Since $P_n$ also returns $u$, we have agreement.

Suppose that the if-condition in $P_n$ evaluates to false. Then two facts are obvious from the implementation: (i) $P_n$ waits until it reads a non-$\perp$ value in *DEC* and returns it, and (ii) the only writes performed on *DEC* are by $P_1, \ldots, P_{n-1}$ in line 5, when they write the return value of $O'$ in *DEC*. Since $O'$ satisfies validity and agreement, it returns the same non-$\perp$ value, say $d$, to all of $P_1, \ldots, P_{n-1}$. Thus, each of $P_1, \ldots, P_{n-1}$ writes $d$ in *DEC* and returns it. $P_n$ eventually reads $d$ in *DEC* and returns it. Hence we have agreement.

Parts 2 and 3 are obvious from the implementation. ◻

**4.3. 1-blocking array implementation.** We now use the group-solo implementation of $n$-consensus to implement, for any constant $c$, an array $\mathcal{O}[1..c]$ of $n$-consensus objects from $(n-1)$-consensus objects and registers. To access the $i$th port of the $j$th object in the array, where $i \in [1..n]$ and $j \in [1..c]$, the implementation provides the access procedure APPLY(*propose* $u, i, \mathcal{O}[j]$). The implementation guarantees some nice properties, but only in certain restricted executions. The implementation treats port $n$ of the array elements differently from ports $1, \ldots, n-1$ in two respects: the restrictions it places on how the port may be used, and the wait-free properties it guarantees for the port.

Specifically, the implementation imposes the following restriction for all ports $i \in [1..n-1]$: port $i$ of two different array elements must not be accessed concurrently.

More precisely, the execution of access procedures at port $i$ of any two array elements $\mathcal{O}[j]$ and $\mathcal{O}[k]$ must not be concurrent. Concurrent execution of access procedures at port $n$ of distinct array elements is, however, permitted.

In executions that satisfy the above restriction, the implementation satisfies validity and agreement and, in addition, guarantees the following two properties: (i) All of $\mathcal{O}[1], \dots, \mathcal{O}[c]$ are wait-free for ports $1, \dots, n-1$, and (ii) all but one of $\mathcal{O}[1], \dots, \mathcal{O}[c]$ are wait-free for port $n$. By property (ii), if access procedures at port $n$ of two different objects in the array are executing concurrently, they do not *both* block. This property will be crucial to the proof of the generalized irreducibility theorem of the next section.

Below we develop the ideas behind the implementation in two stages. In the first stage, we propose an obvious implementation and point out its drawbacks. We fix these drawbacks in the second stage.

*Stage* 1: *Obvious implementation.* Let $GS_1, \dots, GS_c$ be $n$-consensus objects implemented using the group-solo implementation of Figure 3. Access procedure $\text{APPLY}(\textit{propose } u, i, \mathcal{O}[j])$ simply makes a call to $\text{APPLY}(\textit{propose } u, i, GS_j)$.

The drawback is that this implementation fails to satisfy property (ii), stated above. To see this, consider an execution of $\text{APPLY}(\textit{propose } v_1, i_1, \mathcal{O}[1])$, for some $v_1$ and $i_1 \in [1..n-1]$, up to the point where register *GP* of $GS_1$ has been written but before register *DEC* of $GS_1$ has been written (see lines 1 and 5 in Figure 3). Thus, port $n$ of $GS_1$ is now blocked: an access procedure that invokes an operation on port $n$ of $GS_1$ will have to wait until register *DEC* of $GS_1$ is written (see the busy-wait statement in Figure 3). Suppose that, at this point, the access procedure $\text{APPLY}(\textit{propose } v_2, i_2, \mathcal{O}[2])$, for some $v_2$ and $i_2 \in [1..n-1]$ such that $i_2 \neq i_1$, is executed, again up to the point where register *GP* of $GS_2$ has been written but before register *DEC* of $GS_2$ has been written. Thus, port $n$ of $GS_2$ is now blocked as well. If access procedures are now called at port $n$ of $\mathcal{O}[1]$ and $\mathcal{O}[2]$, they will *both* have to wait indefinitely, violating property (ii).

*Stage* 2: *Refined implementation.* We observe that the failure of the above implementation to satisfy property (ii) is due to the fact that it permits port $n$ of more than one *GS* object to become blocked. We now describe a mechanism which ensures that, at all times, port $n$ of at most one *GS* object can be blocked. The basic idea is as follows: When an access procedure $P$ wants to apply *propose* $u$ at port $i \in [1..n-1]$ of $\mathcal{O}[j]$, as in the obvious implementation $P$ applies *propose* $u$ at port $i$ of $GS_j$, but it does so only after completing any propose operations that have already been initiated by other access procedures at ports $1, \dots, n-1$ of $GS_1, \dots, GS_c$. In this way, before $P$ accesses $GS_j$ to perform its own operation (thereby potentially causing port $n$ of that object to become blocked) it ensures that port $n$ of every other object is not blocked.

This idea is implemented as follows. We use $(n-1)$-consensus objects $O_1, \dots, O_c$ (in addition to $GS_1, \dots, GS_c$). When an access procedure $P$ wants to apply an operation *propose* $u$ at a port $i \in [1..n-1]$ of $\mathcal{O}[j]$, $P$ seeks to "obtain permission" to apply *propose* $u$ at port $i$ of $GS_j$. To obtain this permission, $P$ accesses $O_1, \dots, O_c$, in that order, as described below. $P$ proposes the tuple $\langle j, u \rangle$ to $O_1$.[5] Let $\langle k, v \rangle$ be $O_1$'s response. There are two cases: $j \neq k$ or $j = k$. Below, we handle the two cases in turn.

If $j \neq k$, it means that some access procedure has already obtained permission

[5]Strictly speaking, it is natural numbers, and not pairs of natural numbers, that can be proposed to consensus objects. It is well known, however, that pairs of natural number can be "coded" by natural numbers, so this is not a problem.

```
$O_1, \ldots, O_c$: $(n-1)$-consensus objects, initialized to $\perp$
$GS_1, \ldots, GS_c$: $n$-consensus objects, implemented from $(n-1)$-consensus objects and
        registers using the Group-Solo implementation in Figure 3

APPLY(propose $u, i, \mathcal{O}[j]$);  $u \in \mathbb{N}, j \in [1..c]$
                                        $i \in [1..n-1]$
$\ell := 1$
repeat
    $\langle k, v\rangle :=$ APPLY(propose $\langle j, u\rangle, i, O_\ell$)
    $res :=$ APPLY(propose $v, i, GS_k$)
    $\ell := \ell + 1$
until $j = k$
return $res$

APPLY(propose $u, n, \mathcal{O}[j]$);  $u \in \mathbb{N}$,
                                        $j \in [1..c]$
$res :=$ APPLY(propose $u, n, GS_j$)
return $res$
```

FIG. 4. *Implementation of* 1*-blocking array* $\mathcal{O}[1..c]$ *of* $n$*-consensus objects from* $(n-1)$*-consensus objects.*

(from $O_1$) to apply *propose* $v$ to $GS_k$ (at one of its first $n-1$ ports). This operation on $GS_k$ may not have completed, and thus it is possible that port $n$ of $GS_k$ is blocked. $P$ therefore helps complete that operation by applying *propose* $v$ on $GS_k$. $P$ then accesses $O_2$ for permission to apply *propose* $u$ on $GS_j$. It does this by proposing $\langle j, u\rangle$ to $O_2$ and proceeds as above.

If $j = k$, it means that $P$ has the permission to apply a propose operation to $GS_j$. At this point, $P$ can propose either $u$ or $v$ ($u$ is justified since it is $P$'s proposal to $\mathcal{O}[j]$, and $v$ is justified since some access procedure wants to propose it to $\mathcal{O}[j]$). In our implementation, $P$ proposes $v$ to $GS_j$, and regards the response of $GS_j$ as the response of $\mathcal{O}[j]$ to its *propose* $u$ operation.

LEMMA 4.3. *Consider the implementation of array* $\mathcal{O}[1..c]$ *of* $n$*-consensus objects, shown in Figure* 4. *Let* $\mathcal{E}$ *be any execution of this implementation that satisfies the following property:*

(A) *For all ports* $i \in [1..n-1]$ *and all* $j, j' \in [1..c]$ *such that* $j \neq j'$, *the access procedures at port* $i$ *of* $\mathcal{O}[j]$ *and port* $i$ *of* $\mathcal{O}[j']$ *are not concurrent in* $\mathcal{E}$.

*Then, the following hold in* $\mathcal{E}$:

1. *All of* $\mathcal{O}[1], \ldots, \mathcal{O}[c]$ *are wait-free for ports* $1, \ldots, n-1$.
2. *All but one of* $\mathcal{O}[1], \ldots, \mathcal{O}[c]$ *are wait-free for port* $n$.
3. *All of* $\mathcal{O}[1], \ldots, \mathcal{O}[c]$ *satisfy validity and agreement.*

*Proof.* We prove the lemma through a series of claims. The first two claims state that the base objects are accessed as required to ensure they behave properly. As a result, these objects satisfy their safety and liveness properties.

CLAIM 4.3.1. *For each* $i \in [1..n-1]$ *and* $\ell \in [1..c]$ *there are no concurrent operations applied to port* $i$ *of* $O_\ell$ *in* $\mathcal{E}$.

*Proof of Claim* 4.3.1. The claim follows immediately from the following two facts: (i) port $i$ of $O_\ell$ is accessed only by access procedures invoked at port $i$ of $\mathcal{O}[1], \ldots, \mathcal{O}[c]$, and (ii) by (A), access procedures at port $i$ of different objects $\mathcal{O}[j]$ and $\mathcal{O}[j']$ are not concurrent. $\square$

CLAIM 4.3.2. *For each* $i \in [1..n]$ *and* $k \in [1..c]$, *there are no concurrent operations applied to port* $i$ *of* $GS_k$ *in* $\mathcal{E}$.

*Proof of Claim* 4.3.2. For $i \in [1..n-1]$, the claim is immediate from the following two facts: (i) port $i$ of $GS_k$ is accessed only by access procedures invoked at port $i$ of $\mathcal{O}[1], \ldots, \mathcal{O}[c]$, and (ii) by (A), access procedures are not executed concurrently at port $i$ of different objects $\mathcal{O}[j]$ and $\mathcal{O}[j']$. For $i = n$, the claim follows from the observation that port $n$ of $GS_k$ is accessed only by the access procedure at port $n$ of $\mathcal{O}[k]$. ☐

Notice that each proposal to (and hence each response of) $O_\ell$ is of the form $\langle j, u\rangle$. The next claim states that the first components of the responses returned by different objects among $O_1, \ldots, O_c$ are different.

CLAIM 4.3.3. *Let $\langle j, u\rangle$ and $\langle j', u'\rangle$ be the values returned by operations applied to objects $O_\ell$ and $O_{\ell'}$. If $\ell \neq \ell'$, then $j \neq j'$.*

*Proof of Claim* 4.3.3. Suppose $\ell \neq \ell'$. Without loss of generality, we can assume that $\ell' < \ell$. By Claim 4.3.1, $O_\ell$ satisfies validity, and so some access procedure $P$ proposed $\langle j, u\rangle$ to $O_\ell$. By the implementation it is clear that $P$ is an access procedure of the form APPLY(*propose* $u, *, \mathcal{O}[j]$).[6] An inspection of this access procedure shows that before $P$ proposed $\langle j, u\rangle$ to $O_\ell$, it proposed $\langle j, u\rangle$ to $O_{\ell'}$ and received a response $\langle k, v\rangle$, where $k \neq j$. By Claim 4.3.1, $O_{\ell'}$ satisfies agreement, and so $\langle k, v\rangle = \langle j', u'\rangle$. Therefore, $j \neq j'$, as wanted. ☐

CLAIM 4.3.4. *For each $i \in [1..n-1]$ and $j \in [1..c]$, $\mathcal{O}[j]$ is wait-free for port $i$ in $\mathcal{E}$.*

*Proof of Claim* 4.3.4. For each $\ell \in [1..c]$, $O_\ell$ is wait-free for port $i$ since, by Claim 4.3.1, it is accessed properly. By part 2 of Lemma 4.2, $GS_j$ is wait-free for port $i$. Thus, it remains to show that the repeat-loop of access procedure APPLY(*propose* $u$, $i, \mathcal{O}[j]$) in Figure 4 terminates. If that loop does not terminate after $\ell$ iterations, then the propose operations applied to $O_1, \ldots, O_\ell$ return pairs whose first components are different from each other (by Claim 4.3.3) and from $j$. Recall that the first components of propose operations applied to $O_1, \ldots, O_c$ are integers in $[1..c]$, and that these objects satisfy validity since, by Claim 4.3.1, they are accessed properly. Therefore, the loop terminates after at most $c$ iterations. ☐

CLAIM 4.3.5. *All but one of $\mathcal{O}[1], \ldots, \mathcal{O}[c]$ are wait-free for port $n$ in $\mathcal{E}$.*

*Proof of Claim* 4.3.5. Suppose, for contradiction, that there exist $k, k' \in [1..c]$ such that $k \neq k'$ and neither of $\mathcal{O}[k], \mathcal{O}[k']$ is wait-free for port $n$ in $\mathcal{E}$. It follows that neither of $GS_k, GS_{k'}$ is wait-free for port $n$ in $\mathcal{E}$. By part 3 of Lemma 4.2, the following holds for both $GS_k$ and $GS_{k'}$ in $\mathcal{E}$: An operation has been applied to some port other than $n$ and no operation applied to any port returns a response.

Let $P_k$ and $P_{k'}$ denote access procedures that applied, respectively, an operation to a port other than $n$ of $GS_k$ and $GS_{k'}$. From the implementation, it is clear that $P_k$ obtained a response of $\langle k, *\rangle$ from some $O_\ell$ (otherwise $P_k$ would not have applied an operation to $GS_k$). Similarly, $P_{k'}$ obtained a response of $\langle k', *\rangle$ from some $O_{\ell'}$. Since the responses of $O_\ell$ and $O_{\ell'}$ to $P_k$ and $P_{k'}$, respectively, are different (recall that $k \neq k'$), and since each of $O_\ell$ and $O_{\ell'}$ satisfies agreement (recall that, by Claim 4.3.1, these objects are accessed properly), it follows that $\ell \neq \ell'$. Without loss of generality, assume $\ell' < \ell$. From the implementation it is clear that $P_k$ accessed $O_{\ell'}$ before accessing $\mathcal{O}_\ell$. Since $O_{\ell'}$ satisfies agreement, its response to $P_k$ was $\langle k', *\rangle$. It is again clear from the implementation that upon obtaining this response $P_k$ invoked an operation APPLY(*propose* $*, *, GS_{k'}$) and received the corresponding response before proceeding to access $O_{\ell'+1}$. But this contradicts the fact that no operation applied to any port of $GS_{k'}$ returns a response. ☐

CLAIM 4.3.6. *For each $j \in [1..c]$, $\mathcal{O}[j]$ satisfies validity and agreement in $\mathcal{E}$.*

*Proof of Claim* 4.3.6. We observe that the value returned in $\mathcal{E}$ by any of $\mathcal{O}[j]$'s access procedures (namely, APPLY(*propose* $*, *, \mathcal{O}[j]$)) is a response received from

[6]We use an $*$ to denote a quantity whose value is immaterial for the argument at hand.

$GS_j$. By Claim 4.3.2 and Lemma 4.2, $GS_j$ satisfies agreement in $\mathcal{E}$. Therefore, $\mathcal{O}[j]$ also satisfies agreement in $\mathcal{E}$.

Consider any access procedure $P$ of $\mathcal{O}[j]$ that returns $v$ in $\mathcal{E}$. To prove that $\mathcal{O}[j]$ satisfies validity in $\mathcal{E}$ it suffices to show that some access procedure proposed $v$ to $\mathcal{O}[j]$. Since $P$ returns $v$, it is clear from the implementation that $GS_j$ returned $v$ to $P$. By Claim 4.3.2 and Lemma 4.2, $GS_j$ satisfies validity in $\mathcal{E}$. Thus, $v$ was proposed to $GS_j$ by some access procedure $P'$. By the implementation, this implies that one of the following two cases applies:

(i) $P'$ is the access procedure Apply(*propose* $v, n, \mathcal{O}[j]$) and proposes $v$ to $GS_j$.

(ii) $P'$ is an access procedure Apply(*propose* $*, i', \mathcal{O}[j']$), for some $i' \in [1..n-1]$ and $j' \in [1..c]$, and proposes $v$ to $GS_j$ after receiving $\langle j, v\rangle$ from some object $O_\ell$. This means that some access procedure $P''$ proposed $\langle j, v\rangle$ to $O_\ell$, and so $P''$ is an access procedure Apply(*propose* $v, *, \mathcal{O}[j]$).

In either case, some access procedure proposed $v$ to $\mathcal{O}[j]$, as wanted. □

The three parts of Lemma 4.3 are immediate from Claims 4.3.4, 4.3.5, and 4.3.6, respectively. □

## 5. Generalized irreducibility theorem for consensus.

In this section, we prove the generalized irreducibility theorem for consensus, stated as follows. For all $n \geq 2$ and all sets $\mathcal{S}$ of types that include `register`, if there is a wait-free implementation of an $n$-consensus object from $(n-1)$-consensus objects and objects of types in $\mathcal{S}$, then there is a wait-free implementation of an $n$-consensus object just from objects of types in $\mathcal{S}$. In other words, the base $(n-1)$-consensus objects can be eliminated from the implementation. Thus, $(n-1)$-consensus objects are not necessary to implement a wait-free $n$-consensus object, regardless of the base objects that are available for such an implementation.

We obtain this result by repeated application of a lemma stating that if $n$-consensus objects are helpful in implementing $(n+1)$-consensus objects, then $(n-1)$-consensus objects are helpful in implementing $n$-consensus objects.

Lemma 5.1. *For all $n \geq 2$ and all sets $\mathcal{S}$ of types that include* `register`*, if there is a wait-free implementation of an $(n+1)$-consensus object from $n$-consensus objects and objects of types in $\mathcal{S}$, then there is a wait-free implementation of an $n$-consensus object from $(n-1)$-consensus objects and objects of types in $\mathcal{S}$.*

*Proof.* Consider a wait-free implementation of an $(n+1)$-consensus object $\mathcal{O}$ from $n$-consensus objects and objects of types in $\mathcal{S}$. By König's lemma, the number of base objects of $\mathcal{O}$ is finite [2].[7]

Consider any $n$-consensus base object $O$ of $\mathcal{O}$. By Lemma 3.2, we can assume that the binding of $\mathcal{O}$ with $O$ is one-to-one static. Since $\mathcal{O}$ has $n+1$ ports and $O$ has $n$ ports, some port of $\mathcal{O}$, say $p$, does not use any port of $O$. Let $i_1, i_2, \ldots, i_n$ be the remaining ports of $\mathcal{O}$ (i.e., the elements of $[1..n+1] - \{p\}$) listed in ascending order. We may assume without loss of generality (by renaming ports of $O$, if necessary) that ports $i_1, i_2, \ldots, i_n$ of $\mathcal{O}$ are bound, respectively, to ports $1, 2, \ldots, n$ of $O$. We may therefore classify the base objects of $\mathcal{O}$ into four categories:

(a) $A_1, \ldots, A_a$ are the $n$-consensus base objects that are not accessed by port $n+1$ of $\mathcal{O}$. Thus, ports $n-1$ and $n$ of $A_1, \ldots, A_a$ are bound, respectively, to ports $n-1$ and $n$ of $\mathcal{O}$.

(b) $B_1, \ldots, B_b$ are the $n$-consensus base objects that are not accessed by port $n$ of $\mathcal{O}$. Thus, ports $n-1$ and $n$ of $B_1, \ldots, B_b$ are bound, respectively, to ports

[7]It is here that we make use of the assumption that types exhibit finite nondeterminism.

$n-1$ and $n+1$ of $\mathcal{O}$.

(c) $C_1, \ldots, C_c$ are the $n$-consensus base objects that are not accessed by one of the first $n-1$ ports of $\mathcal{O}$. Thus, ports $n-1$ and $n$ of $C_1, \ldots, C_c$ are bound, respectively, to ports $n$ and $n+1$ of $\mathcal{O}$.

(d) $D_1, \ldots, D_d$ are the remaining base objects of $\mathcal{O}$ (these belong to the types in $\mathcal{S}$).

Modify the implementation of $\mathcal{O}$ as follows:

- Replace the base objects $A_1, \ldots, A_a$ and $B_1, \ldots, B_b$ with a 1-blocking array $O[1..a+b]$, implemented as in Figure 4. Objects $O[1..a]$ of the array are used in the place of $A_1, \ldots, A_a$ and $O[a+1..a+b]$ are used in the place of $B_1, \ldots, B_b$.
- Replace the base objects $C_1, \ldots, C_c$ with $NC_1, \ldots, NC_c$, where each $NC_i$ is implemented using the nonconcurrent implementation in Figure 2.
- The access procedures of the new implementation are the same as in the original implementation with the following exception: Each time any access procedure Apply(*propose* $*, *, \mathcal{O}$) applies, in the original implementation, a *propose* $u$ operation to port $i$ of $A_j$, $B_j$, or $C_j$, the new implementation requires the access procedure instead to apply *propose* $u$ to port $i$ of $O[j]$, $O[a+j]$, or $NC_j$, respectively.

Claim 5.1.1. *The* $(n+1)$*-consensus object* $\mathcal{O}$ *in the new implementation described above has the following properties:*

1. *All base consensus objects of* $\mathcal{O}$ *are* $(n-1)$*-consensus objects and all other base objects of* $\mathcal{O}$ *belong to types in* $\mathcal{S}$.
2. $\mathcal{O}$ *satisfies validity and agreement in all executions in which operations are not executed concurrently at ports* $n-1$ *and* $n$ *of* $NC_i$ *for all* $i \in [1..c]$.
3. $\mathcal{O}$ *is wait-free for ports* $1, \ldots, n-1$.
4. $\mathcal{O}$ *is wait-free for one of ports* $n$ *and* $n+1$.

*Proof of Claim* 5.1.1. Part 1 follows from the fact that $O[1..a+b]$ and $NC_1, \ldots, NC_c$ are implemented from $(n-1)$-consensus objects and registers, the latter being in $\mathcal{S}$ by assumption.

To prove part 2, consider any execution $\mathcal{E}$ of the new implementation of $\mathcal{O}$ in which operations are not executed concurrently at ports $n-1$ and $n$ of any $NC_i$, $i \in [1..c]$. The following three facts imply that $\mathcal{O}$ satisfies validity and agreement in $\mathcal{E}$:

- The original implementation of $\mathcal{O}$ satisfies validity and agreement in all executions.
- By part 3 of Lemma 4.3, $O[1], \ldots, O[a+b]$ (which have replaced $A_1, \ldots, A_a$ and $B_1, \ldots, B_b$ of the original implementation) satisfy validity and agreement in all executions that satisfy the following proviso: there are no concurrent invocations at the *same* port $p \in [1..n-1]$ of two *distinct* objects $O[j]$ and $O[k]$, where $j, k \in [1..a+b]$. This proviso is satisfied in $\mathcal{E}$ because (i) the first $n-1$ ports of each object $O[1], \ldots, O[a+b]$ are bound to the first $n-1$ ports of $\mathcal{O}$, respectively; and (ii) in the original implementation of $\mathcal{O}$, no access procedure invokes concurrent operations on distinct base objects—and, in particular, on the $n$-consensus base objects $A_1, \ldots, A_a$ and $B_1, \ldots, B_b$ that were replaced by $O[1], \ldots, O[a+b]$.
- By part 2 of Lemma 4.1, $NC_1, \ldots, NC_c$ (which have replaced $C_1, \ldots, C_c$ of the original implementation) satisfy validity and agreement in $\mathcal{E}$.

Part 3 follows from the following facts: (i) $NC_1, \ldots, NC_c$ are wait-free (by part 1 of Lemma 4.1), (ii) $O[1], \dots, O[a+b]$ are wait-free for ports $1, \dots, n-1$ (by part 1 of Lemma 4.3), and (iii) none of the first $n-1$ ports of $\mathcal{O}$ use port $n$ of any of $O[1], \dots, O[a+b]$.

Part 4 follows from the following facts: (i) all but one of $O[1], \dots, O[a+b]$ are wait-free for port $n$ (by part 2 of Lemma 4.3), and (ii) port $n$ of each of $O[1], \dots, O[a]$ is used only by port $n$ of $\mathcal{O}$, and port $n$ of each of $O[a+1], \dots, O[a+b]$ is used only by port $n+1$ of $\mathcal{O}$ (thus, port $n$ of each $O[i]$ is used either by port $n$ of $\mathcal{O}$ or by port $n+1$ of $\mathcal{O}$, but not by both). $\square$

Next we describe how to transform the new implementation of $\mathcal{O}$, the $(n+1)$-consensus object satisfying the properties in Claim 5.1.1, into an implementation of an $n$-consensus object $\mathcal{O}'$. Informally, the access procedure for each of the first $n-1$ ports of $\mathcal{O}'$ simply calls the access procedure at the corresponding port of $\mathcal{O}$ and returns that procedure's response. For port $n$, the access procedure of $\mathcal{O}'$ executes the access procedures of *both* ports $n$ and $n+1$ of $\mathcal{O}$, alternating between the two in such a manner that one of them is guaranteed to terminate and return a value; that value then becomes the response of the access procedure of port $n$ of $\mathcal{O}'$.

To explain more precisely how $\mathcal{O}'$ works, we need to make some observations and introduce some terminology. Recall (see Figure 4) that to propose a value to port $n$ of an object $O[j]$ in a 1-blocking array, we simply propose the same value to port $n$ of an object $GS_j$ in the group-solo implementation. If, in some execution, the latter enters the busy-wait statement (see Figure 3), we say that object $O[j]$ is *blocked*.

Here, now, is how the access procedure APPLY(*propose* $u, i, \mathcal{O}'$) works:

- For all $i \in [1..n-1]$, APPLY(*propose* $u, i, \mathcal{O}'$) executes APPLY(*propose* $u, i, \mathcal{O}$).
- APPLY(*propose* $u, n, \mathcal{O}'$), the access procedure at port $n$ of $\mathcal{O}'$, interleaves the execution of APPLY(*propose* $u, n, \mathcal{O}$) and APPLY(*propose* $u, n+1, \mathcal{O}$), using the rules below. For convenience, we let PROC$_n$ and PROC$_{n+1}$ denote APPLY(*propose* $u, n, \mathcal{O}$) and APPLY(*propose* $u, n+1, \mathcal{O}$), respectively. Note that, by construction of $\mathcal{O}$, PROC$_n$ applies operations to port $n$ of $O[1], \dots, O[a]$ (as well as to port $n-1$ of $NC_1, \dots, NC_c$, and possibly to ports of $D_1, \dots, D_d$). Similarly, PROC$_{n+1}$ applies operations to port $n$ of $O[a+1], \dots, O[a+b]$ (as well as to port $n$ of $NC_1, \dots, NC_c$, and possibly to ports of $D_1, \dots, D_d$).
  - (a) Begin by executing PROC$_n$.
  - (b) Suspend PROC$_n$ and resume PROC$_{n+1}$ if and only if PROC$_n$ accesses an object $O[j]$ that is blocked (for some $j \in [1..a]$). Similarly, suspend PROC$_{n+1}$ and resume PROC$_n$ if and only if PROC$_{n+1}$ accesses an object $O[a+j]$ that is blocked (for some $j \in [1..b]$).
  - (c) As soon as either PROC$_n$ or PROC$_{n+1}$ terminates and returns some value $v$, terminate APPLY(*propose* $u, n, \mathcal{O}'$) and return $v$.

CLAIM 5.1.2. *The $n$-consensus object $\mathcal{O}'$, in the implementation described above, has the following properties:*

1. *All base consensus objects of $\mathcal{O}'$ are $(n-1)$-consensus objects, and all other base objects belong to types in $\mathcal{S}$.*
2. *$\mathcal{O}'$ satisfies validity and agreement.*
3. *$\mathcal{O}'$ is wait-free.*

*Proof of Claim* 5.1.2. Part 1 follows directly from part 1 of Claim 5.1.1.

Each access procedure of $\mathcal{O}'$ merely returns the value of a corresponding access procedure of $\mathcal{O}$. By part 2 of Claim 5.1.1, $\mathcal{O}$ satisfies validity and agreement as long as there are no concurrent operations at ports $n-1$ and $n$ of each $NC_i$ for $i \in [1..c]$.

Thus, to prove part 2 of the present claim, it suffices to prove that, for every $i \in [1..c]$, the access procedures of $\mathcal{O}'$ do not apply concurrent operations to ports $n-1$ and $n$ of $NC_i$. To see why this is the case, recall that ports $n-1$ and $n$ of each $NC_i$ are accessed only by $\text{PROC}_n$ and $\text{PROC}_{n+1}$, respectively. By construction of $\mathcal{O}'$, only the access procedure of port $n$ of $\mathcal{O}'$ executes $\text{PROC}_n$ and $\text{PROC}_{n+1}$. Furthermore, while the access procedure of port $n$ of $\mathcal{O}'$ is executing one of $\text{PROC}_n$ or $\text{PROC}_{n+1}$, it has suspended execution of the other. Therefore, no concurrent operations are executed at ports $n-1$ and $n$ of $NC_i$, as wanted.

Part 3 of Claim 5.1.1 implies that $\mathcal{O}'$ is wait-free for ports $1, \ldots, n-1$, and part 4 of Claim 5.1.1 implies that $\mathcal{O}'$ is wait-free for port $n$. Thus, $\mathcal{O}'$ is wait-free for all its $n$ ports. □

This completes the proof of Lemma 5.1. □

THEOREM 5.2 (generalized irreducibility theorem for consensus). *For all $n \geq 2$ and all sets $\mathcal{S}$ of types that include* `register`*, if there is a wait-free implementation of an $n$-consensus object from $(n-1)$-consensus objects and objects of types in $\mathcal{S}$, then there is a wait-free implementation of an $n$-consensus object from objects of types in $\mathcal{S}$.*

*Proof.* Suppose there is a wait-free implementation of an $n$-consensus object from $(n-1)$-consensus objects and objects of types in $\mathcal{S}$. By repeated application of Lemma 5.1 we have that for all $k \in [2..n]$, there is a wait-free implementation of a $k$-consensus object from $(k-1)$-consensus objects and objects of types in $\mathcal{S}$. Composing all these implementations, we obtain a wait-free implementation of an $n$-consensus object from 1-consensus objects and objects of types in $\mathcal{S}$. Since 1-consensus objects have a trivial implementation,[8] we have a wait-free implementation of an $n$-consensus object from objects of types in $\mathcal{S}$. □

**6. Equivalence of $t$-resilient and wait-free implementations of consensus.** In this section, we prove that the three versions of fault-tolerant consensus—wait-free consensus, weak $t$-tolerant consensus, and strong $t$-tolerant consensus—are equivalent: if any of them can be implemented from certain types of shared objects, the other two can also be implemented from the same types of shared objects. More precisely, we have the following.

THEOREM 6.1 (equivalence of $t$-resilient and wait-free consensus). *For all $n > t \geq 2$ and all sets $\mathcal{S}$ of types that include* `register`*, the following three statements are equivalent:*

S1. *There is an implementation of a weakly $t$-resilient $n$-consensus object from objects of types in $\mathcal{S}$.*

S2. *There is a wait-free implementation of a $(t+1)$-consensus object from objects of types in $\mathcal{S}$.*

S3. *There is an implementation of a strongly $t$-resilient $n$-consensus object from objects of types in $\mathcal{S}$.*

We prove the theorem by showing that S1 implies S2, S2 implies S3, and S3 implies S1. As we will see, the generalized irreducibility theorem of the previous section is used to prove the first of these three claims.

LEMMA 6.2. S1 *implies* S2*, where statements* S1 *and* S2 *are as in Theorem* 6.1.

*Proof.* Let $\mathcal{O}$ be a $t$-resilient implementation of an $n$-consensus object from objects of types in $\mathcal{S}$. Using $\mathcal{O}$ (and a few other base objects) we implement a wait-free $(t+1)$-consensus object $\mathcal{O}'$. Roughly speaking, the access procedures of $\mathcal{O}'$ coordinate with

[8] A *propose $u$* operation on a 1-consensus object simply returns $u$.

each other to simulate an execution of $\mathcal{O}$. The simulation is done in such a way that each access procedure of $\mathcal{O}'$ that crashes can prevent *at most one* access procedure of $\mathcal{O}$ in the simulated execution from making progress. Thus, if access procedures in at most $t$ ports of $\mathcal{O}'$ crash, then access procedures of at most $t$ ports of $\mathcal{O}$ will stop making progress in the simulated execution. Since $\mathcal{O}$ is $t$-resilient, the access procedures in the remaining $n - t$ ports of $\mathcal{O}$ eventually terminate and return the same value. This value is adopted as the return value of $\mathcal{O}'$.

We implement the above idea as follows:

- We employ registers $U_1, \ldots, U_n$ and $R_1, \ldots, R_n$. For each $i \in [1..n]$, $U_i$ stores the value proposed at port $i$ of $\mathcal{O}$, and $R_i$ stores the current state of that port's access procedure. (The state of an access procedure consists of the values of all its private variables and of its "program counter.")
- We employ $(t+1)$-ported test&set objects $TS_1, \ldots, TS_n$. (The test&set object type has two states—*win* and *lose*—and supports two operations: *test&set* and *reset*. The *test&set* operation returns the current state of the object and sets the state to *lose*. The *reset* sets the state to *win* and returns an acknowledgment as a response.)
  We use $TS_j$ to ensure that at any time at most one of $\mathcal{O}'$'s access procedures simulates steps of the access procedure at port $j$ of $\mathcal{O}$.
- For brevity, let $P_i$ denote the access procedure Apply(*propose* $u, i, \mathcal{O}'$) for each $i \in [1..t+1]$. $P_i$ considers the $n$ ports of $\mathcal{O}$ in round-robin fashion. When $P_i$ considers port $j$ of $\mathcal{O}$, it applies a *test&set* operation to $TS_j$. If $TS_j$ returns *lose*, $P_i$ moves on to consider the next port of $\mathcal{O}$, $\big((j+1) \mod n\big)+1$. If $TS_j$ returns *win*, $P_i$ advances the simulation of the access procedure at port $j$ of $\mathcal{O}$ by performing the following actions:
  (1) $P_i$ reads $R_j$ to determine the current state of that procedure.
  (2) If no step of port $j$ has been simulated before, $P_i$ writes its proposal $u$ into $U_j$ and sets $R_j$ to the initial state of Apply(*propose* $U_j, j, \mathcal{O}$).
  (3) $P_i$ performs a single step of Apply(*propose* $U_j, j, \mathcal{O}$) and writes the resulting state of that procedure in $R_j$.
  (4) $P_i$ performs a reset operation on $TS_j$ (so that some $P_{i'}$, $i' \in [1..t+1]$, may execute the next step of the access procedure at $\mathcal{O}$'s port $j$).
  (5) If the step that $P_i$ simulated caused the access procedure of port $j$ of $\mathcal{O}$ to terminate and return $v$, $P_i$ writes $v$ in a register *DEC* and terminates; otherwise, $P_i$ moves on to consider the next port of $\mathcal{O}$.

The implementation of $\mathcal{O}'$, described informally above, is shown in Figure 5. The following claim states the desired properties of $\mathcal{O}'$.

Claim 6.2.1. *The* $(t+1)$*-consensus object* $\mathcal{O}'$*, shown in Figure* 5*, satisfies validity and agreement and is wait-free.*

*Proof of Claim* 6.2.1. Let $\mathcal{E}'$ be an arbitrary concurrent execution of $\mathcal{O}'$'s access procedures, and let $\mathcal{E}$ be the subexecution of $\mathcal{E}'$ consisting of the operations applied to $\mathcal{O}$ (see line 8 in Figure 5). Thus $\mathcal{E}$ is a concurrent execution of $\mathcal{O}$'s access procedures. To prove the claim it suffices to show that $\mathcal{O}'$ satisfies validity and agreement and is wait-free in $\mathcal{E}'$.

It is clear from the implementation that the values returned by access procedures of $\mathcal{O}'$ in $\mathcal{E}'$ are values returned by access procedures of $\mathcal{O}$ in $\mathcal{E}$. In addition, any value proposed by an access procedure of $\mathcal{O}$ in $\mathcal{E}$ is a value proposed by some access procedure of $\mathcal{O}'$ in $\mathcal{E}'$ (see line 6). From these two observations, and the fact that $\mathcal{O}$ satisfies validity and agreement in $\mathcal{E}$, it follows that $\mathcal{O}'$ satisfies validity and agreement in $\mathcal{E}'$.

```
        $\mathcal{O}$: $t$-resilient implementation of $n$-consensus object, initialized to $\bot$
        $TS_1, \ldots, TS_n$: $(t+1)$-ported test&set objects, initialized to $win$
        $R_1, \ldots, R_n$: registers, initialized to $\bot$
        $U_1, \ldots, U_n$: registers, initialized arbitrarily
        $DEC$: register, initialized to $\bot$
        $\underline{\text{APPLY}(propose\ u, i, \mathcal{O}')}$; $u \in \mathbb{N}, i \in [1..t+1]$

1.      $j := 1$
2.      repeat
3.          if APPLY($test\&set, i, TS_j$) = $win$ then
4.              read $R_j$ to determine the state of PROC$_j$
5.              if $R_j = \bot$ then
6.                  $U_j := u$
7.                  $R_j :=$ initial state of APPLY($propose\ U_j, j, \mathcal{O}$)
8.              execute one step of APPLY($propose\ U_j, j, \mathcal{O}$)
9.              write in $R_j$ the new state of APPLY($propose\ U_j, j, \mathcal{O}$)
10.             if APPLY($propose\ U_j, j, \mathcal{O}$) terminated and returned $v$ then $DEC := v$
11.             APPLY($reset, i, TS_j$)
12.         if $DEC \neq \bot$ then return $DEC$
13.         $j := \big((j+1) \mod n\big) + 1$
14.     forever
```

FIG. 5. *Wait-free implementation of a $(t+1)$-consensus object $\mathcal{O}'$ from t-resilient implementation of n-consensus object $\mathcal{O}$.*

It remains to show that $\mathcal{O}'$ is wait-free in $\mathcal{E}'$. Suppose, for contradiction, that there is an access procedure, say $P$, of $\mathcal{O}'$ that is infinite in $\mathcal{E}'$. For $i \in [1..t+1]$ let $P_i$ denote the access procedure at port $i$ of $\mathcal{O}'$ in $\mathcal{E}'$, and for $j \in [1..n]$ let $Q_j$ denote the access procedure at port $j$ of $\mathcal{O}$ in $\mathcal{E}$. We make a subclaim that, for each finite $Q_j$ in $\mathcal{E}$ ($j \in [1..n]$), there is a $P_i$ ($i \in [1..t+1]$) such that $P_i$ applies only finitely many operations to $TS_j$ in $\mathcal{E}'$, the last of which is a *test&set* that returns *win*. We prove this subclaim by contradiction: suppose that every $P_i$ that receives *win* from $TS_j$ at line 3 subsequently resets it at line 11. Then, since $P$ applies infinitely many *test&set* operations to $TS_j$ in $\mathcal{E}'$, there would be infinitely many *test&set* operations to $TS_j$ that return *win* in $\mathcal{E}'$. As a result, either infinitely many steps of $Q_j$ are performed in $\mathcal{E}$ or $Q_j$ completes in $\mathcal{E}$ and returns some response $v$. The former case contradicts the premise that $Q_j$ is finite in $\mathcal{E}$. In the latter case, whichever $P_k$ simulated the last step of $Q_j$ would write $v$ in *DEC* (at line 10) before resetting $TS_j$. After this writing in *DEC*, when $P$ reads *DEC* at line 12, it finds $v$ in *DEC* and therefore completes, which contradicts the premise that $P$ is infinite. This completes the proof of the subclaim. It is clear from Figure 5 that if $Q_j$ and $Q_{j'}$ are distinct access procedures of $\mathcal{O}$ that are finite in $\mathcal{E}$, then the corresponding access procedures of $\mathcal{O}'$ that executed the last *test&set* operation on $TS_j$ and $TS_{j'}$, respectively, are also distinct. Thus, it is immediate from the subclaim that if at most $m$ of $P_1, P_2, \ldots, P_{t+1}$ are finite, then at most $m$ of $Q_1, \ldots, Q_n$ are finite. Since, by our supposition, $P$ is infinite (and $P$ is one of $P_1, \ldots, P_{t+1}$), it follows that at most $t$ of $Q_1, \ldots, Q_n$ are finite, which implies that at least $n-t$ of $Q_1, \ldots, Q_n$ are infinite. This conclusion contradicts the fact that $\mathcal{O}$ is $t$-resilient. Hence, we conclude that $\mathcal{O}'$ is wait-free in $\mathcal{E}'$. ∎

Afek, Weisberger, and Weisman showed that there is a wait-free implementation of a $k$-ported test&set object from 2-consensus objects and registers for all $k \geq 2$ [1].

This, together with Claim 6.2.1, implies the following.

CLAIM 6.2.2. *There is a wait-free implementation of a* $(t+1)$*-consensus object from* 2*-consensus objects and objects of types in* $\mathcal{S}$*.*

We now have all the ingredients needed to complete the proof of Lemma 6.2. Since $t \geq 2$, Claim 6.2.2 implies that there is a wait-free implementation of a $(t+1)$-consensus object from $t$-consensus objects and objects of types in $\mathcal{S}$. This, together with Theorem 5.2, implies that there is a wait-free implementation of a $(t+1)$-consensus object from just objects of types in $\mathcal{S}$. This completes the proof of Lemma 6.2. ◻

The next lemma has a similar proof, so we provide only an informal sketch.

LEMMA 6.3. S2 *implies* S3*, where statements* S2 *and* S3 *are as in Theorem* 6.1*.*

*Proof.* The proof is based on a simulation of a strong $t$-resilient $n$-consensus object $\mathcal{O}'$ using a wait-free $(t+1)$-consensus object $\mathcal{O}$. This simulation is similar to that in the proof of Lemma 6.2 (see Figure 5). The only difference is that now a larger number $n$ of access procedures simulate the steps of a wait-free implementation for a smaller number $t+1$ of access procedures (in Figure 5 it is the other way around: a smaller number $t+1$ of access procedures simulate the steps of a $t$-resilient implementation for a larger number $n$ of access procedures). In this way, even if only a few access procedures of the $t$-resilient implementation are active, they will nevertheless simulate an execution of the wait-free implementation and reach a decision that satisfies validity and agreement. As before, the simulation uses a register *DEC* (to publish the decision value) and test&set objects to ensure that the simulation is done correctly. (One test&set object is used at each port $i$ of $\mathcal{O}$ to ensure that at most one access procedure of $\mathcal{O}'$ simulates port $i$ of $\mathcal{O}$ at any time.)

The above argument shows that a strong $t$-resilient $n$-consensus object $\mathcal{O}'$ can be simulated using a wait-free $(t+1)$-consensus object $\mathcal{O}$, $t+1$ $n$-ported test&set objects, and the register *DEC*. We show next that the test&set objects can be eliminated from the simulation. As mentioned earlier, an $n$-ported test&set object can be implemented wait-free from 2-consensus objects and registers [1]. Since $t \geq 2$, it follows that $n$-ported test&set objects used in the simulation can be substituted by their implementations from $(t+1)$-consensus objects and registers. With this substitution, we have a simulation of a strong $t$-resilient $n$-consensus object $\mathcal{O}$ from registers and a set of wait-free $(t+1)$-consensus objects. By the premise of the lemma (i.e., statement S2), we have that (1) a wait-free $(t+1)$-consensus object can be implemented from objects that belong to the types in $\mathcal{S}$, and (2) $\mathcal{S}$ includes the register type. It follows that a strong $t$-resilient $n$-consensus object can be simulated using only objects whose types are in $\mathcal{S}$. ◻

The next lemma trivially holds since strong $t$-resilience implies weak $t$-resilience.

LEMMA 6.4. S3 *implies* S1*, where statements* S3 *and* S1 *are as in Theorem* 6.1*.*

Theorem 6.1 is immediate from Lemmas 6.2, 6.3, and 6.4.

Notice that the equivalence of statements S1, S2, and S3 in the theorem is proved for $t \geq 2$. As shown by Lo and Hadzilacos, the implication S1 $\implies$ S2 breaks down for $t = 1$ [22]. The other implications, however, continue to hold for $t = 1$: S2 implies S3 (the proof is the same as in Lemma 6.3) and S3 implies both S1 and S2 (by definition).

Finally, we present some corollaries of the equivalence theorem. Herlihy proved that there is no wait-free implementation of 3-consensus from objects belonging to any of the following sets of types [12]: $\{\texttt{queue}, \texttt{register}\}$, $\{\texttt{stack}, \texttt{register}\}$, $\{\texttt{fetch\&add}, \texttt{register}\}$, $\{\texttt{swap}, \texttt{register}\}$. (Here, the types `register`, `queue`, `stack`, `fetch&add`, and `swap` can have *any* number of ports; the impossibility result holds regardless.) This, together with Theorem 6.1, implies the following.

COROLLARY 6.5. *For all $n \geq 3$, there is no 2-resilient implementation of an n-consensus object from objects belonging to any of the following sets of types:* {`queue`, `register`}, {`stack`, `register`}, {`fetch&add`, `register`}, {`swap`, `register`}.

Consider a MEM($m$) object that corresponds to Herlihy's $m$-register assignment memory [12]. Informally, MEM($m$) consists of an infinite array of cells; it supports the *read* $i$ operation, which returns the value in the $i$th cell, and the *write*$(i_1, v_1, \ldots, i_m, v_m)$ operation, which (atomically) writes $v_j$ in cell $i_j$ for all $j \in [1..m]$. For $m \geq 2$, Herlihy proved that there is no wait-free implementation of a $(2m-1)$-consensus object from MEM($m$) objects (regardless of the number of ports the MEM($m$) objects may have). This, together with Theorem 6.1, implies the following.

COROLLARY 6.6. *For all $m \geq 2$ and $n \geq 2m-1$, there is no $(2m-2)$-resilient implementation of an n-consensus object from* MEM($m$) *objects.*

**Acknowledgments.** We thank Michael Merritt for many helpful suggestions and the anonymous referees for their comments on an earlier draft of this paper.

## REFERENCES

[1] Y. AFEK, E. WEISBERGER, AND H. WEISMAN, *A completeness theorem for a class of synchronization objects*, in Proceedings of the 12th Annual ACM Symposium on Principles of Distributed Computing, 1993, pp. 159–170.

[2] R. BAZZI, G. NEIGER, AND G. PETERSON, *On the use of registers in achieving wait-free consensus*, in Proceedings of the 13th Annual ACM Symposium on Principles of Distributed Computing, 1994, pp. 354–363.

[3] E. BOROWSKY AND E. GAFNI, *Generalized FLP impossibility result for t-resilient asynchronous computations*, in Proceedings of the 25th ACM Symposium on Theory of Computing, 1993, pp. 91–100.

[4] E. BOROWSKY AND E. GAFNI, *The Implication of the Borowsky-Gafni Simulation on the Set Consensus Hierarchy*, Technical Report 930021, Computer Science Department, UCLA, 1993.

[5] E. BOROWSKY, E. GAFNI, AND Y. AFEK, *Consensus power makes (some) sense*, in Proceedings of the 13th Annual ACM Symposium on Principles of Distributed Computing, 1994, pp. 363–372.

[6] E. BOROWSKY, E. GAFNI, N. LYNCH, AND S. RAJSBAUM, *The BG distributed simulation algorithm*, Distributed Comput., 14 (2001), pp. 127–146.

[7] T. CHANDRA, V. HADZILACOS, P. JAYANTI, AND S. TOUEG, *Wait-freedom vs. t-resiliency and the robustness of wait-free hierarchies*, in Proceedings of the 13th Annual ACM Symposium on Principles of Distributed Computing, 1994, pp. 334–343.

[8] S. CHAUDHURI AND P. REINERS, *Understanding the set consensus partial order using the Borowsky and Gafni simulation*, in Proceedings of the 10th International Workshop on Distributed Algorithms (WDAG), Lecture Notes in Comput. Sci. 1151, Springer-Verlag, New York, 1996, pp. 362–379.

[9] B. CHOR, A. ISRAELI, AND M. LI, *Wait-free consensus using asynchronous hardware*, SIAM J. Comput., 23 (1994), pp. 701–712.

[10] D. DOLEV, C. DWORK, AND L. STOCKMEYER, *On the minimal synchronism needed for distributed consensus*, J. ACM, 34 (1987), pp. 77–97.

[11] M. FISCHER, N. LYNCH, AND M. PATERSON, *Impossibility of distributed consensus with one faulty process*, J. ACM, 32 (1985), pp. 374–382.

[12] M. P. HERLIHY, *Wait-free synchronization*, ACM Trans. Programming Languages and Systems, 13 (1991), pp. 124–149.

[13] M. P. HERLIHY AND N. SHAVIT, *The asynchronous computability theorem for t-resilient tasks*, in Proceedings of the 25th ACM Symposium on Theory of Computing, 1993, pp. 111–120.

[14] M. P. HERLIHY AND J. M. WING, *Linearizability: A correctness condition for concurrent objects*, ACM Trans. Programming Languages and Systems, 12 (1990), pp. 463–492.

[15] P. JAYANTI, *Wait-free computing*, in Proceedings of the 9th International Workshop on Distributed Algorithms (WDAG), Lecture Notes in Comput. Sci. 972, Springer-Verlag, New York, 1995, pp. 19–50.

[16] P. Jayanti, *Robust wait-free hierarchies*, J. ACM, 44 (1997), pp. 592–614.

[17] P. Jayanti and S. Toueg, *Some results on the impossibility, universality, and decidability of consensus*, in Proceedings of the 6th International Workshop on Distributed Algorithms (WDAG), Lecture Notes in Comput. Sci. 647, Springer-Verlag, New York, 1992, pp. 69–84.

[18] L. Lamport, *Concurrent reading and writing*, Comm. ACM, 20 (1977), pp. 806–811.

[19] L. Lamport, *On interprocess communication.* I: *Basic formalism*, Distributed Comput., 1 (1986), pp. 77–85.

[20] L. Lamport, *On interprocess communication.* II: *Algorithms*, Distributed Comput., 1 (1986), pp. 86–101.

[21] W.-K. Lo and V. Hadzilacos, *All of us are smarter than any of us: Nondeterministic wait-free hierarchies are not robust*, SIAM J. Comput., 30 (2000), pp. 689–728.

[22] W. Lo and V. Hadzilacos, *On the power of shared object types to implement one-resilient consensus*, Distributed Comput., 13 (2000), pp. 219–238.

[23] W. K. Lo, *More on t-resilience vs. wait-freedom*, in Proceedings of the 14th Annual ACM Symposium on Principles of Distributed Computing, 1995, pp. 110–119.

[24] M. Loui and H. Abu-Amara, *Memory requirements for agreement among unreliable asynchronous processes*, in Advances in Computing Research, Vol. 4, JAI Press, Greenwich, CT, 1987, pp. 163–183.

[25] N. Lynch and M. Tuttle, *An Introduction to Input/Output Automata*, Tech. Report MIT/LCS/TM-373, Laboratory for Computer Science, MIT, Cambridge, MA, 1988.

[26] S. Moran and L. Rappoport, *On the robustness of* $\mathtt{h}_{\mathtt{m}}^{\mathtt{r}}$, in Proceedings of the 10th International Workshop on Distributed Algorithms (WDAG), Lecture Notes in Comput. Sci. 1151, Springer-Verlag, New York, 1996, pp. 344–361.

[27] G. Peterson, R. Bazzi, and G. Neiger, *A gap theorem for consensus types*, in Proceedings of the 13th Annual ACM Symposium on Principles of Distributed Computing, 1994, pp. 344–353.

[28] G. L. Peterson, *Concurrent reading while writing*, ACM Trans. Programming Languages and Systems, 5 (1983), pp. 56–65.

[29] M. Saks and F. Zaharoglou, *Wait-free k-set agreement is impossible: The topology of public knowledge*, in Proceedings of the 25th ACM Symposium on Theory of Computing, 1993, pp. 101–110.

[30] E. Schenk, *Computability and Complexity Results for Agreement Problems in Shared Memory Distributed Systems*, Ph.D. thesis, University of Toronto, 1996.

SIAM J. COMPUT.
Vol. 34, No. 2, pp. 358–372

# APPROXIMATION ALGORITHMS FOR THE 0-EXTENSION PROBLEM*

GRUIA CALINESCU†, HOWARD KARLOFF‡, AND YUVAL RABANI§

**Abstract.** In the 0-*extension problem*, we are given a weighted graph with some nodes marked as *terminals* and a semimetric on the set of terminals. Our goal is to assign the rest of the nodes to terminals so as to minimize the sum, over all edges, of the product of the edge's weight and the distance between the terminals to which its endpoints are assigned. This problem generalizes the multiway cut problem of Dahlhaus et al. [*SIAM J. Comput.*, 23 (1994), pp. 864–894] and is closely related to the metric labeling problem introduced by Kleinberg and Tardos [*Proceedings of the* 40*th IEEE Annual Symposium on Foundations of Computer Science*, New York, 1999, pp. 14–23].

We present approximation algorithms for 0-EXTENSION. In arbitrary graphs, we present a $O(\log k)$-approximation algorithm, $k$ being the number of terminals. We also give $O(1)$-approximation guarantees for weighted planar graphs. Our results are based on a natural *metric relaxation* of the problem previously considered by Karzanov [*European J. Combin.*, 19 (1998), pp. 71–101]. It is similar in flavor to the linear programming relaxation of Garg, Vazirani, and Yannakakis [*SIAM J. Comput.*, 25 (1996), pp. 235–251] for the multicut problem, and similar to relaxations for other graph partitioning problems. We prove that the integrality ratio of the metric relaxation is at least $c\sqrt{\lg k}$ for a positive $c$ for infinitely many $k$. Our results improve some of the results of Kleinberg and Tardos, and they further our understanding on how to use metric relaxations.

**Key words.** metric space, approximation algorithm, linear programming relaxation, graph partitioning

**AMS subject classification.** 68W25

**DOI.** 10.1137/S0097539701395978

## 1. Introduction.

Let $V$ be a finite set, let $T \subseteq V$, and let $d$ be a semimetric on $T$.[1] Then a semimetric $\delta$ on $V$ is an *extension of $d$ to $V$* iff for every $i, j \in T$, $\delta(i,j) = d(i,j)$. If, in addition, for every $i \in V$ there exists $j \in T$ such that $\delta(i,j) = 0$, then $\delta$ is a 0-*extension of $d$ to $V$*.

We consider the following optimization problem, denoted 0-EXTENSION and posed by Karzanov [13]: Given a clique $V$ with a nonnegative edge weight $c(e)$ for every edge $e$, a subset $T$ of the nodes, and a semimetric $d$ on $T$, find a 0-extension $\delta$ of $d$ to $V$ that minimizes $\sum_{uv\in E} c(u,v)\delta(u,v)$.

Before doing anything else, we give an alternate formulation of 0-EXTENSION: Given the input above, find a function $f : V \to T$ such that $f(t) = t$ for all $t \in T$ which

*Received by the editors October 2, 2001; accepted for publication (in revised form) July 20, 2004; published electronically December 1, 2004.
http://www.siam.org/journals/sicomp/34-2/39597.html

†Department of Computer Science, Illinois Institute of Technology, Chicago, IL 60616 (calinesc@cs.iit.edu). This author's research was performed while at DIMACS and Georgia Institute of Technology, and supported in part by a DIMACS postdoctoral fellowship and by NSF grant CCR-9732746.

‡AT&T Labs–Research, Room C231, 180 Park Ave., Florham Park, NJ 07932 (howard@research.att.com). This author's research was performed in part while at Georgia Institute of Technology and supported in part by NSF grant CCR-9732746.

§Computer Science Department, Technion–IIT, Haifa 32000, Israel (rabani@cs.technion.ac.il). This author's work at the Technion was supported by BSF grant 96-00402, by Ministry of Science contract 9480198, by grant 386/99-1 of the Israel Science Foundation founded by the Israeli Academy of Sciences and Humanities, through the CONSIST consortium by the Ministry of Trade and Industry's MAGNET program, and by the Fund for the Promotion of Research at the Technion.

[1]A function $d : T \times T \to \mathbb{R}$ is a *semimetric* on $T$ iff for every $i_1, i_2, i_3 \in T$, $d(i_1, i_1) = 0$, $d(i_1, i_2) \geq 0$, $d(i_1, i_2) = d(i_2, i_1)$, and $d(i_1, i_2) + d(i_2, i_3) \geq d(i_1, i_3)$. If, in addition, $d(i_1, i_2) = 0$ implies $i_1 = i_2$, then $d$ is a *metric*.

minimizes $\sum_{uv\in E} c(u,v)d(f(u),f(v))$. It is easy to see that the two formulations are equivalent. For given a feasible solution of the first kind, we can define $f(u)$ to be some terminal $i$ such that $\delta(u,i)=0$, choosing $f(u)=u$ if $u\in T$, and given a feasible solution of the second kind, we can define $\delta(u,v)=d(f(u),f(v))$ for all $u,v\in V$; the costs of the two solutions are identical because if $u,v\in V$, $i,j\in T$, and $\delta(u,i)=\delta(v,j)=0$, then $\delta(u,v)=\delta(i,j)=d(i,j)$. Often, instead of defining the edge weights on *all* edges of a clique on $V$, we will define $c(u,v)$ for each edge $uv$ in a given graph $G=(V,E)$, where $c(u,v)=0$ for $uv\notin E$ is assumed. That way, we can exploit the structure of $G$ if, say, $G$ is planar.

It helps to compare 0-Extension to the multiway cut problem of Dahlhaus et al. [7, 8]. Multiway Cut is the following problem: Given a graph $G=(V,E)$ with nonnegative edge weights $c:E\to\mathbb{R}$, and a subset $T\subseteq V$ of terminals, find a mapping $f:V\to T$, such that $f(t)=t$ for all $t\in T$, so as to minimize

$$\sum_{uv\in E, f(u)\neq f(v)} c(u,v).$$

In other words, find a set of edges of minimum total weight whose removal disconnects all terminal pairs. If we define $d$ to be the uniform metric on $T$, i.e., $d(i,j)=1$ if $i\neq j$ and $d(i,i)=0$, then Multiway Cut is exactly this problem: Find $f:V\to T$, with $f(t)=t$ for all $t\in T$, so as to minimize

$$\sum_{uv\in E} c(u,v)\cdot d(f(u),f(v)),$$

as $d(f(u),f(v))=1$ if $f(u)\neq f(v)$ and $d(f(u),f(v))=0$ otherwise. Now 0-Extension is the natural generalization of Multiway Cut in which, instead of being the uniform metric, $d$ is an arbitrary semimetric on $T$. In other words, we must find an $f:V\to T$, with $f(t)=t$ for all $t\in V$, so as to minimize

$$\sum_{uv\in E} c(u,v)\cdot d(f(u),f(v)).$$

Dahlhaus et al. [8] show that Multiway Cut (and therefore 0-Extension) is APX-hard. Thus there exists a constant $\alpha>1$ such that no polynomial-time algorithm can find a solution within a factor of $\alpha$ of the optimum, unless P=NP.

In this paper we develop approximation algorithms for the 0-extension problem. We study what seems to us to be the most natural linear programming relaxation for the 0-extension problem: Find a minimum weight extension of $d$ to $V$; specifically, given the semimetric $d$ on $T$, extend $d$ to a semimetric $\delta$ on the larger set $V$ so as to minimize $\sum_{uv\in E} c(u,v)\delta(u,v)$. (We call this the *metric relaxation.*) Obviously, the set of feasible extensions $\delta$ is defined by $O(|V|^3)$ linear constraints, and the objective function is linear. Thus finding the best extension is a linear programming problem, and thus it can be solved in polynomial time. We derive approximation algorithms using the metric relaxation, thus bounding also the integrality ratio for the relaxation. For arbitrary graphs we give a randomized $O(\log|T|)$-approximation algorithm. We show that the integrality ratio is at least a constant times $\sqrt{\log|T|}$ for infinitely many $|T|$. We improve the upper bounds to $O(1)$ for (weighted) planar graphs (or, in fact, for any family of graphs that excludes a $K_{r,r}$-minor for some fixed $r$).

Karzanov [13] considers the metric relaxation for the 0-extension problem and characterizes some of the cases in which the relaxation gives the optimal solution. For the multiway cut problem, it was known that the metric relaxation has integrality ratio exactly $2 - 2/|T|$ [5, Theorem 3.1]. Indeed, this observation uses the same idea underlying the combinatorial algorithm of Dahlhaus et al. [8] that has the same performance guarantee. For the general case, the quality of the metric relaxation was not known prior to our work. For multiway cut, a different relaxation gives better approximations, with an asymptotic ratio significantly below 2 (see Calinescu, Karloff, and Rabani [2] and the improved bounds of Cunningham and Tang [6] and of Karger et al. [12]). It is not clear whether the Calinescu, Karloff, and Rabani relaxation can be extended to handle the general case of 0-Extension.

In a recent paper, Kleinberg and Tardos [16] give approximation algorithms for a similar problem of classification with pairwise relations, which they call Metric Labeling. In their problem, the terminals are distinct from the vertices and are called *labels*. There is a semimetric on the labels, and for each node of the graph there is a vector of assignment costs to each of the labels. The goal is to minimize the total assignment cost plus the sum of weighted edge lengths. More formally, given a graph $G = (V, E)$ with nonnegative edge weights $c : E \to \mathbb{R}$, a set $T$ of labels, a semimetric $d$ on $T$, and a nonnegative assignment cost function $a : V \times T \to \mathbb{R} \cup \{\infty\}$, Metric Labeling is the problem of finding a mapping $f : V \to T$ so as to minimize

$$\sum_{u \in V} a(u, f(u)) + \sum_{uv \in E} c(u, v) \cdot d(f(u), f(v)).$$

For the case in which $d$ is the uniform metric, Kleinberg and Tardos give a 2-approximation algorithm, based on a relaxation similar to the Calinescu et al. relaxation for Multiway Cut. The integrality ratio for the relaxation here, as opposed to the relaxation for the multiway cut problem, is at least $2 - 2/|T|$. Chuzhoy [4] improves their result for three and four labels (achieving a tight 4/3 bound for three labels). Kleinberg and Tardos further give a constant approximation algorithm for a class of tree metrics, the so-called hierarchically well-separated tree metrics (HST metrics). Following Bartal's small distortion embeddings of metrics into HST metrics [1], they use a constant-ratio approximation algorithm for HST metrics to give an $O(\log |T| \log \log |T|)$-approximation algorithm for arbitrary metrics. Gupta and Tardos [11] later give a local search-based 4-approximation algorithm for the case that $d$ is a truncated linear metric (i.e., $T = \{1, 2, \ldots, k\}$ and for some value $m$, $d(i, j) = \min\{|i - j|, m\}$). Recently, Chekuri et al. [3] introduced a new linear programming relaxation for the metric labeling problem, and using it, obtained another $O(\log |T| \log \log |T|)$-approximation algorithm for arbitrary metrics and a $2 + \sqrt{2}$-approximation algorithm for the truncated linear metric.

The authors of [16] and [11] motivate their work by several applications, mostly concerning computer vision, such as image restoration and visual correspondence. In these applications the nodes of the graph are pixels in a raster image, and the edges model adjacency (in fact, the graph is a two-dimensional mesh). In image restoration applications, the labels model pixel intensities or colors. Assigning a label to a pixel amounts to determining the "true" intensity (or color) of the pixel from the observed values. The assignment cost penalizes for the difference between the observed and assigned intensity. In visual correspondence applications, the labels model possible shifts between two images. Assigning a label to a pixel amounts to determining the shift of that pixel between the two images. The assignment cost penalizes for the difference between the values of the supposedly matching pixels. In both types of applications, the structure of the graph arises from assuming that the a priori distribution of "true" labels is generated by a Markov random field (where the distribution of a pixel depends only on the distribution of its neighbors).

Note that 0-Extension is a special case of Metric Labeling: Given an instance of 0-Extension with $T \subseteq V$, define $a : V \times T \to \mathbb{R} \cup \{\infty\}$ by

- if $u \in T$, then $a(u,u) = 0$ and $a(u,t) = \infty$ for all $t \in T \setminus \{u\}$;
- if $u \notin T$, then $a(u,t) = 0$ for all $t \in T$.

The feasible solutions to Metric Labeling of finite cost then correspond to functions $f : V \to T$, which are arbitrary except for the constraint that $f(t) = t$ for all terminals t, and the objective function value corresponding to $f$ is $\sum c(u,v)d(f(u), f(v))$, the value of the objective function of 0-Extension.

Thus, our results improve upon the results in [16] for this case. We note that the 0-extension formulation seems appropriate for many of the computer vision applications mentioned in [16, 11]. For example, if we connect each pixel by a weighted edge to the label corresponding to its observed intensity, we get an assignment cost proportional to the distance between the observed and assigned value. Our algorithm for weighted planar graphs actually assumes only that $V \setminus T$ induces a planar graph. Thus we get a constant-ratio approximation algorithm for some of these computer vision problems, for an arbitrary metric on the labels $T$.

Another problem related to ours is the multicut problem, first considered in the context of approximation algorithms in two papers by Garg, Vazirani, and Yannakakis [9, 10] (and implicitly in Klein et al. [14]). In this problem, we are given a (weighted) graph and $k$ *pairs* of terminals (nodes in the graph), and the goal is to find a minimum weight set of edges whose removal disconnects every pair of terminals. This is a different generalization of multiway cut (the latter can be viewed as the multicut problem for all $\binom{k}{2}$ pairs of terminals). It is not comparable to the 0-extension problem, in the sense that neither problem is a special case of the other. In [10], Garg et al. give an $O(\log |T|)$ approximation algorithm for the multicut problem, based on a metric relaxation which assigns lengths to edges so that the distance between every specified pair of terminals is at least 1. Their result is tight for the relaxation. The example achieving (asymptotically) the integrality ratio is an expander. For their upper bounds, they use a region-growing technique similar to that used by Leighton and Rao [17] for approximating the minimum flux (edge expansion) of a graph. Klein, Plotkin, and Rao [15] improve the Leighton and Rao technique for planar graphs (and more generally for graphs that exclude a $K_{r,r}$-minor) to get a constant factor approximation for the minimum flux. Using their technique, Tardos and Vazirani [18] exhibit a constant-factor approximation algorithm for the multicut problem on planar graphs (and $K_{r,r}$-minor free graphs).

Our result can be seen as a counterpart to the Garg et al. and the Tardos and Vazirani results. The region-growing technique does not give a good approximation in the case of 0-extension. However, our results can be viewed as a form of (randomized) region growing after the application of a scaling function to the distances. This is implicit in the general case algorithm, and explicit in the planar graphs algorithm, where we use the Klein et al. technique on the scaled distances. It is worth noting that, in contrast with the situation of [10], expanders are *not* a particularly bad case for our relaxation (see section 4).

The rest of the paper is organized as follows. In section 2 we present the algorithm for the general case. Section 3 has the improved bounds for planar graphs. In section 4 we discuss the quality of the linear programming relaxation underlying our approximation algorithms. Throughout the rest of the paper we use $k$ to denote the number $|T|$ of terminals. We call the vertices in $V \setminus T$ *nonterminals*.

## 2. An $O(\log k)$-approximation algorithm.

In this section we present the randomized algorithm which finds a 0-extension of weight at most $O(\log k)$ times the optimum. We begin by computing an optimal solution to the following natural linear programming relaxation, which we denote by (MET):

$$\text{Minimize } \sum_{uv \in E} c(u,v)\delta(u,v) \quad \text{subject to}$$

$$(V,\delta) \text{ is a semimetric,} \tag{2.1}$$

$$\delta(i,j) = d(i,j) \qquad \forall i,j \in T. \tag{2.2}$$

If an assignment $f : V \to T$ defines an optimal solution to the 0-extension problem, then setting $\delta(u,v) = d(f(u), f(v))$ defines a feasible solution of (MET) of the same weight as the optimal solution. Therefore, the optimal value $Z^*$ of (MET) is a lower bound on the minimum weight 0-extension. We exhibit a rounding procedure that takes any feasible solution $\delta$ of (MET) of value $Z$ and constructs a 0-extension assignment $f : V \to T$ whose weight is $O(Z \log k)$.

Our rounding procedure works as follows. Pick uniformly at random a permutation $\sigma$ of $T$ and independently choose, uniformly at random in the interval $[1,2)$, a real number $\alpha$. The rounding algorithm iteratively assigns some nodes to terminal $\sigma_1$, then some of the remaining nodes to terminal $\sigma_2$, and so on. For every $u \in V$, set $A_u = \min_{i \in T} \delta(u,i)$. The rounding procedure is given below.

Rounding Procedure.

```
Set $f(t) = t$ for all terminals $t$.
Pick a random permutation $\sigma = \langle \sigma_1, \sigma_2, \ldots, \sigma_k \rangle$ of the terminals.
Pick $\alpha$ uniformly at random in the interval $[1,2)$.
for $j = 1$ to $k$ do
    for all unassigned nonterminals $u$ such that $\delta(u,\sigma_j) \le \alpha A_u$, do
        Set $f(u) = \sigma_j$ (i.e., assign $u$ to $\sigma_j$).
    endfor
endfor
```

We first show that the rounding procedure produces a 0-extension, as follows.

Claim 2.1. *The rounding procedure assigns every nonterminal to a terminal.*

*Proof.* Consider a nonterminal $v$, and let $t \in T$ be a terminal with $\delta(v,t) = A_u$. Choose $j$ such that $t = \sigma_j$. Then if $v$ is not assigned to a terminal in iterations $1, 2, \ldots, j-1$, it must be assigned to $t$ in iteration $j$, because $\alpha \ge 1$. ☐

For any pair of nodes $u, v \in V$, define a random variable $s(u,v) = d(f(u), f(v))$. We say that $u, v \in V$ are *separated* if $f(u) \ne f(v)$. Note that if $u, v$ are not separated, then $s(u,v) = 0$. Our goal is to bound the expectation $E[s(u,v)]$. We first state a bound on the probability that $u, v$ are separated.

Lemma 2.2. *Fix $u, v \in V$ and let $\delta = \delta(u,v)$. If $0 < \delta \le \frac{1}{4}\min\{A_u, A_v\}$, then*

$$\Pr[u, v \textit{ are separated}] \le 4\mathcal{H}_k \left( \frac{\delta}{A_u} + \frac{\delta}{A_v} \right),$$

*where $\mathcal{H}_k = 1 + \frac{1}{2} + \cdots + \frac{1}{k}$ is the kth harmonic number.*

Before we prove this bound, we state and prove its consequence, a bound for $E[s(u,v)]$, which is the core of the analysis of our algorithm.

LEMMA 2.3. *For any pair of distinct vertices $u, v \in V$, $E[s(u,v)] \le 38\mathcal{H}_k\delta(u,v)$.*[2]

*Proof.* Fix $u \neq v$ and set $\delta = \delta(u,v)$. By the triangle inequality, $A_v \le A_u + \delta$ and $A_u \le A_v + \delta$. We have $s(u,v) = d(f(u), f(v)) = \delta(f(u), f(v)) \le \delta(f(u), u) + \delta(u,v) + \delta(v, f(v))$. As $\alpha \in [1,2)$, we obtain

$$s(u,v) \le 2A_u + \delta + 2A_v. \tag{2.3}$$

If $u$ is a terminal, then $A_u = 0$ and $A_v \le \delta$. Therefore, by inequality (2.3), $s(u,v) \le 3\delta$, regardless of the choice of $\sigma$ and $\alpha$. If both $u$ and $v$ are nonterminals and $\delta = 0$, then by the triangle inequality, for any terminal $j \in T$, $\delta(u,j) = \delta(v,j)$. Therefore, $u$ and $v$ are assigned to the same terminal, regardless of the choice of $\sigma$ and $\alpha$, so $s(u,v) = 0 = \delta$.

Thus we may assume that both $u$ and $v$ are nonterminals, and that $\delta > 0$. We consider two cases, depending on whether or not $\delta$ is small compared to $A_u$ or $A_v$. If $A_u < 4\delta$, then $A_v < 5\delta$, and by inequality (2.3), $s(u,v) < 2(4+5)\delta + \delta = 19\delta$. Similarly, if $A_v < 4\delta$, then $s(u,v) < 19\delta$. Therefore, if $A_u < 4\delta$ or $A_v < 4\delta$, the lemma holds.

Assume, therefore, that $\delta \le \frac{1}{4}\min\{A_u, A_v\}$. (Recall that we also assume that $\delta > 0$.) We have

$$\begin{aligned} E[s(u,v)] &\le 4\mathcal{H}_k\left(\frac{\delta}{A_u} + \frac{\delta}{A_v}\right)(2A_u + 2A_v + \delta) \\ &\le 4\mathcal{H}_k\delta\left(\frac{4A_u + 3\delta}{A_u} + \frac{4A_v + 3\delta}{A_v}\right) \\ &\le 4\mathcal{H}_k\delta\left(4 + \frac{3}{4} + 4 + \frac{3}{4}\right) = 38\mathcal{H}_k\delta, \end{aligned}$$

where the first inequality follows from Lemma 2.2 and inequality (2.3). ◻

We conclude with the following result.

THEOREM 2.4. *There is a randomized polynomial-time $O(\log k)$-approximation algorithm for* 0-EXTENSION.

*Proof.* Let $\delta^*$ be an optimal solution of (MET) of cost $Z^*$. By Lemma 2.3, the expected weight of the 0-extension obtained by the rounding procedure is $O(Z^* \log k)$. Therefore, there exists a choice of $\sigma$ and of $\alpha$ that produces a solution of weight $O(Z^* \log k)$. To obtain a polynomial-time algorithm, notice that for a given permutation $\sigma$, two different values of $\alpha$, $\alpha_1 > \alpha_2$, produce combinatorially distinct solutions only if there is a terminal $j$ and a node $u$ such that $\delta^*(u,j) \le \alpha_1 A_u$ but $\delta^*(u,j) > \alpha_2 A_u$. Thus we can enumerate over at most $k|V|$ "interesting" values of $\alpha$. We can determine these values by sorting the fractions $\delta^*(u,j)/A_u$, over all nodes $u$ with $A_u > 0$ and over all $j \in T$. ◻

*Proof of Lemma* 2.2. Let $\mathcal{E}(u,v)$ denote the event that there is a terminal $j$ such that when $j$ is processed $u$ is assigned to $j$ whereas $v$ remains unassigned; define $\mathcal{E}(v,u)$ similarly. We will show that

$$\Pr[\mathcal{E}(u,v)] \le \frac{4\mathcal{H}_k\delta}{A_u}. \tag{2.4}$$

By symmetry, $\Pr[\mathcal{E}(v,u)] \le 4\mathcal{H}_k\delta/A_v$. Therefore, the lemma follows from inequality (2.4).

[2]The constant 38 is somewhat arbitrary, and definitely could be improved.

Label the $k$ terminals so that $\delta(u,1) \leq \delta(u,2) \leq \cdots \leq \delta(u,k)$. For $j = 1,2,\ldots,k$, let $l_j = \delta(u,j)/A_u$; $1 = l_1 \leq l_2 \leq l_3 \leq \cdots \leq l_k$. Let $r_j = \delta(v,j)/A_v \geq 1$.

For $\gamma \geq 1$, let

$$M(\gamma) = \{j \in T \mid l_j \leq \gamma < r_j\},$$

and let

$$S(\gamma) = \{j \in T \mid \gamma \geq r_j\}.$$

Note that $M(\gamma)$ and $S(\gamma)$ are disjoint subsets of terminals. Set $m(\gamma) = |M(\gamma)|$ and $s(\gamma) = |S(\gamma)|$.

Let $\alpha \in [1,2)$ be the value used by the rounding procedure. Then $v$ must be assigned to a terminal in $S(\alpha)$, because $\delta(v,j) \leq \alpha A_v$ is equivalent to $r_j \leq \alpha$. Similarly, $u$ cannot be assigned to a terminal which is not in $M(\alpha) \cup S(\alpha)$. Indeed, $u$ can only be assigned to a terminal $j$ with $l_j \leq \alpha$, and based on whether $r_j \leq \alpha$ or not, $j$ is either in $S(\alpha)$ or in $M(\alpha)$. Moreover, $\mathcal{E}(u,v)$ happens iff the first terminal in $M(\alpha) \cup S(\alpha)$ to be processed is in $M(\alpha)$. Indeed, if the first such terminal is $j \in S(\alpha)$, then $v$ (and possibly also $u$) will be assigned to $j$. Thus

$$\Pr[\mathcal{E}(u,v) \mid \alpha] = \frac{m(\alpha)}{m(\alpha)+s(\alpha)}.$$

(Note that there exists a $j$ for which $r_j = 1$, and thus $S(\alpha)$ is never empty.) As $\alpha$ is distributed uniformly in $[1,2)$, we get

$$\Pr[\mathcal{E}(u,v)] = \int_1^2 \frac{m(\alpha)}{m(\alpha)+s(\alpha)} d\alpha. \tag{2.5}$$

We need the following claim.

CLAIM 2.5. *Fix a positive integer $k$ and a nonnegative real $\beta$. Let $(\langle l_1,l_2,\ldots,l_k\rangle, \langle r_1,r_2,\ldots,r_k\rangle)$ be a pair of real sequences such that $1 = l_1 \leq l_2 \leq l_3 \leq \cdots \leq l_k$, $r_j \geq 1$ for all $j$, some $r_j = 1$, and (either $r_j - l_j \leq \beta$ or $l_j > 2$) for all $j$. Define functions $m : [1,\infty) \to \mathbb{Z}$ and $s : [1,\infty) \to \mathbb{Z}$ as follows: $m(\alpha) = |\{j \mid l_j \leq \alpha < r_j\}|$ and $s(\alpha) = |\{j \mid \alpha \geq r_j\}|$. Then*

$$\int_1^2 \frac{m(\alpha)}{m(\alpha)+s(\alpha)} d\alpha \leq \mathcal{H}_k \beta.$$

*Proof.* Note that $s(\alpha) \geq 1$ for all $\alpha$ (because some $r_j = 1$), so the function we are integrating is well defined. Let $t$ be the largest index for which $l_t \leq 2$. We prove by induction on $t$ that the value of the integral is at most $\mathcal{H}_t\beta \leq \mathcal{H}_k\beta$. For $t = 1$ the claim holds, because for $\alpha \in [r_1, 2]$, $m(\alpha) = 0$. As $r_1 - 1 = r_1 - l_1 \leq \beta$, we get

$$\int_1^2 \frac{m(\alpha)}{m(\alpha)+s(\alpha)} d\alpha \leq \int_1^{\min\{r_1,2\}} 1 d\alpha \leq r_1 - 1 \leq \beta = \mathcal{H}_1\beta.$$

Assume that the claim is true for the case in which exactly $t-1$ $j$'s satisfy $l_j \leq 2$ for some $t \geq 2$, and consider pairs $(\langle l_1,l_2,\ldots,l_k\rangle, \langle r_1,r_2,\ldots,r_k\rangle)$ in which exactly $t$ $j$'s are such that $l_j \leq 2$. We compare the value $I$ of the integral in this case to the value $I'$ of the integral for the pair in which the $t$th coordinate of the first sequence is replaced by $l_{t+1}$, except that if $t = k$, the $t$th coordinate is replaced by 3. We use

$m'(\alpha)$ and $s'(\alpha)$ for the latter pair, where clearly $s'(\alpha) = s(\alpha)$ for all $\alpha$. Note that the latter pair satisfies the hypotheses of the claim, and, furthermore, only $t-1$ $j$'s are such that $l_j \le 2$. Therefore, by the inductive hypothesis, $I' \le \mathcal{H}_{t-1}\beta$.

If $l_t \ge r_t$, then for every $\alpha \in (1,2)$, $m'(\alpha) = m(\alpha)$, and therefore $I = I'$, thus establishing the claim in this case. Thus we may assume that $l_t < r_t$. Clearly $m(\alpha) \in \{m'(\alpha), m'(\alpha)+1\}$ for any $\alpha$. Therefore, for any $\alpha \in (1,2)$,

$$\begin{aligned}\frac{m(\alpha)}{m(\alpha)+s(\alpha)} - \frac{m'(\alpha)}{m'(\alpha)+s'(\alpha)} &\le \frac{m'(\alpha)+1}{m'(\alpha)+1+s'(\alpha)} - \frac{m'(\alpha)}{m'(\alpha)+s'(\alpha)}\\ &\le \frac{1}{m'(\alpha)+s'(\alpha)+1}.\end{aligned}$$

Now $m(\alpha) \ne m'(\alpha)$ implies $\alpha \in [l_t, r_t]$, and $\alpha \in [l_t, r_t]$ implies $l_j \le l_t \le \alpha$ for all $j \le t-1$. However, every $j \le t-1$ satisfying $l_j \le \alpha$ satisfies either $l_j \le \alpha < r_j$ or $\alpha \ge r_j$, and hence each such $j$ contributes to at least one of $m'(\alpha)$ and $s'(\alpha)$. Hence $\alpha \in [l_t, r_t]$ implies $m'(\alpha)+s'(\alpha) \ge t-1$ and $1/(m'(\alpha)+s'(\alpha)+1) \le 1/t$.

Therefore $I - I' \le (r_t - l_t)(1/t)$. Because $l_t \le 2$, $r_t - l_t \le \beta$. Hence, $I \le I' + \beta/t \le \mathcal{H}_{t-1}\beta + \beta/t = \mathcal{H}_t\beta$. This completes the proof of Claim 2.5. ∎

We now proceed with the proof of Lemma 2.2. Note that if $l_j \le 2$, then $\delta(u,j) \le 2A_u$. Using the assumption that $\delta \le \frac{1}{4}A_u$, we have, for such $j$,

$$\begin{aligned} r_j - l_j &\le \frac{\delta(u,j)+\delta}{A_v} - \frac{\delta(u,j)}{A_u}\\ &\le \frac{\delta(u,j)+\delta}{A_u-\delta} - \frac{\delta(u,j)}{A_u} = \frac{\delta(\delta(u,j)+A_u)}{A_u(A_u-\delta)}\\ &\le \frac{3\delta}{A_u-\delta}\\ &\le \frac{4\delta}{A_u}.\end{aligned}$$

Hence, using Claim 2.5 with $\beta = 4\delta/A_u$, we have

$$\int_1^2 \frac{m(\alpha)}{m(\alpha)+s(\alpha)}d\alpha \le \mathcal{H}_k \cdot \frac{4\delta}{A_u}. \tag{2.6}$$

Combining (2.5) and inequality (2.6), we get $\Pr[\mathcal{E}(u,v)] \le 4\mathcal{H}_k\delta/A_u$, which proves inequality (2.4) and thus the lemma. ∎

## 3. An $O(1)$-approximation algorithm for planar graphs.

In this section we use the linear programming relaxation (MET) to get improved bounds for planar graphs. To achieve the improved bounds, we present a different rounding procedure. We show that if the input graph $G = (V,E)$ does not have a $K_{r,r}$-minor, then the rounding procedure presented in this section guarantees an $O(r^3)$ approximation ratio. As planar graphs are $K_{3,3}$-minor free, this gives a polynomial-time $O(1)$-approximation algorithm for planar graphs (and, more generally, for $K_{r,r}$-minor free graphs, for every fixed $r$).

The main tool that we use is the following theorem of Klein, Plotkin, and Rao [15] (the extension to the weighted case was stated by Tardos and Vazirani [18]).

THEOREM 3.1 (see Klein, Plotkin, and Rao). *There are constants $\kappa$ and $\lambda$ and a polynomial-time algorithm $KPR(H,\delta,c,\gamma,r)$ which takes as input a graph $H = (V_H, E_H)$ with nonnegative integral edge lengths $\delta : E_H \to \mathbb{Z}$ and nonnegative edge* *costs $c : E_H \to \mathbb{Q}$, a positive rational $\gamma$, and a positive integer $r$, and which finds either (1) a $K_{r,r}$-minor in $H$ or (2) a set of edges of total $c$-cost at most $\kappa \frac{r}{\gamma} \sum_{e \in E_H} \delta(e)c(e)$ whose removal decomposes $H$ into connected components called* clusters *such that the shortest path (in $H$, using edge lengths $\delta$) between any two nodes in the same component is at most $\lambda r^2 \gamma$.* □

Let $r$ be a positive integer. Let $\delta : V \times V \to \mathbb{R}$ be a feasible solution of (MET) of weight $Z$. Using Theorem 3.1, we exhibit a deterministic rounding procedure that obtains a 0-extension of weight $O(Z)$, assuming that the input graph $G$ is $K_{r,r}$-minor–free.

The main idea of the rounding procedure is to partition the nonterminals into clusters such that, for any two nodes $u$ and $v$ in the same cluster, $A_u$ is at most twice $A_v$. We then assign all the nodes in a cluster to a terminal closest to one of the nodes in the cluster. More formally, the rounding procedure computes a 0-extension $f : V \to T$ as follows.

Second Rounding Procedure.

```
Set $f(t) = t$ for every terminal $t$.
**for** every nonterminal $u \in V$ such that $A_u = 0$, **do**
    Set $f(u) \leftarrow i$ for some $i \in T$ with $\delta(u,i) = 0$.
**endfor**
Let $\bar{G} = (\bar{V}, \bar{E})$ be the subgraph of $G$ induced by the remaining nonterminals.
$\bar{\delta}_{\min} \leftarrow \min\{\delta(u,v)/\max\{A_u, A_v\} \mid uv \in \bar{E},\ \delta(u,v) > 0\}$.
**for** every edge $uv \in \bar{E}$, **do**
    $\tilde{\delta}(uv) \leftarrow \lceil \delta(u,v)/(\bar{\delta}_{\min} \cdot \max\{A_u, A_v\}) \rceil$,
    $\tilde{c}(uv) \leftarrow c(u,v) \cdot \max\{A_u, A_v\}$,
**endfor**
Execute KPR$(\bar{G}, \tilde{\delta}, \tilde{c}, 1/(2\lambda r^2 \bar{\delta}_{\min}), r)$.
**for** each resulting cluster $C \subseteq \bar{V}$, **do**
    Choose $x \in C$ to minimize $A_x$.
    Choose $i \in T$ such that $\delta(x,i) = A_x$.
    Set $f(u) \leftarrow i$ for all $u \in C$.
**endfor**
```

We first establish a few simple facts about this rounding procedure. Let $\tilde{Z} = \sum_{uv \in \bar{E}} \tilde{\delta}(uv)\tilde{c}(uv)$.

Claim 3.2. $\tilde{Z} \le 2Z/\bar{\delta}_{\min}$.

*Proof.* Note that if $\delta(u,v) > 0$, then $\delta(u,v)/(\bar{\delta}_{\min} \cdot \max\{A_u, A_v\}) \ge 1$; hence $\tilde{\delta}(uv) = \lceil \delta(u,v)/(\bar{\delta}_{\min} \cdot \max\{A_u, A_v\}) \rceil \le 2\delta(u,v)/(\bar{\delta}_{\min} \cdot \max\{A_u, A_v\})$. We have

$$\begin{aligned}
\sum_{uv \in \bar{E}} \tilde{\delta}(uv)\tilde{c}(uv) &= \sum_{uv \in \bar{E}} \left\lceil \frac{\delta(u,v)}{\bar{\delta}_{\min} \cdot \max\{A_u, A_v\}} \right\rceil \cdot (c(u,v) \cdot \max\{A_u, A_v\}) \\
&\le \frac{2}{\bar{\delta}_{\min}} \sum_{uv \in \bar{E}} \delta(u,v)c(u,v) \\
&\le \frac{2}{\bar{\delta}_{\min}} \sum_{uv \in E} \delta(u,v)c(u,v) \\
&= 2Z/\bar{\delta}_{\min}. \qquad \square
\end{aligned}$$

Claim 3.3. *The total $\tilde{c}$-cost of the edges removed by KPR$(\bar{G}, \tilde{\delta}, \tilde{c}, 1/(2\lambda r^2 \bar{\delta}_{\min}), r)$ is at most $4\kappa\lambda r^3 Z$, where $\kappa$ and $\lambda$ are the constants from Theorem* 3.1*. Moreover, each of the resulting clusters $C$ has $\tilde{\delta}$-diameter at most $1/(2\bar{\delta}_{\min})$.*

*Proof.* By Theorem 3.1, the sum of $\tilde{c}(uv)$ over edges $uv$ with $u, v$ in different clusters is at most $(\kappa r/\gamma)\tilde{Z}$. By Claim 3.2, this is at most $4\kappa\lambda r^3 Z$ (since $\gamma = 1/(2\lambda r^2 \bar{\delta}_{\min})$). Also, by Theorem 3.1, the $\tilde{\delta}$-diameter of each resulting cluster $C$ is at most $1/(2\bar{\delta}_{\min})$. ◻

We now relate the $\tilde{\delta}$-distances to the original $\delta$-distances.

LEMMA 3.4. *Let $u, v \in \bar{V}$. If the length of a shortest path in $\bar{G}$ between $u$ and $v$ with respect to edge lengths $\tilde{\delta}$ is at most $1/(2\bar{\delta}_{\min})$, then $\delta(u,v) \le A_u$.*

*Proof.* Let $\langle u = x_0, x_1, \ldots, x_j = v\rangle$ be a shortest path in $\bar{G}$ between $u$ and $v$ with respect to the edge lengths $\tilde{\delta}$. For $1 \le t \le j$, let $s_t = \sum_{i=1}^{t} \delta(x_{i-1}, x_i)$. By the triangle inequality, $A_{x_t} \le A_u + \sum_{i=1}^{t} \delta(x_{i-1}, x_i) = A_u + s_t$. Note that $s_1 \le s_2 \le \cdots \le s_j$. Therefore, for $i \le j$, $\delta(x_{i-1}, x_i)/\max\{A_{x_{i-1}}, A_{x_i}\} \ge \delta(x_{i-1}, x_i)/(A_u + s_i) \ge \delta(x_{i-1}, x_i)/(A_u + s_j)$. Also, $\tilde{\delta}(x_{i-1}, x_i) \ge \delta(x_{i-1}, x_i)/(\max\{A_{x_{i-1}}, A_{x_i}\}\bar{\delta}_{\min})$, and therefore $\delta(x_{i-1}, x_i)/\max\{A_{x_{i-1}}, A_{x_i}\} \le \bar{\delta}_{\min}\tilde{\delta}(x_{i-1}, x_i)$. Using this, we have $s_j/(A_u + s_j) = \sum_{i=1}^{j} \delta(x_{i-1}, x_i)/(A_u + s_j) \le \bar{\delta}_{\min} \sum_{i=1}^{j} \tilde{\delta}(x_{i-1}, x_i) \le \bar{\delta}_{\min}/(2\bar{\delta}_{\min}) = 1/2$, where the last inequality follows from the hypothesis that the length of $\langle x_0, x_1, \ldots, x_j\rangle$ is at most $1/(2\bar{\delta}_{\min})$. We conclude that $s_j \le A_u$. Finally, notice that, by the triangle inequality, $\delta(u,v) \le s_j$. ◻

We are ready to analyze the performance of the rounding procedure.

LEMMA 3.5. *Let $r > 0$ be an integer. Then for every input graph $G$ which is $K_{r,r}$-minor–free, for every feasible solution to (MET) of weight $Z$, the above rounding procedure produces a 0-extension of weight at most $(4 + 16\kappa\lambda r^3)Z$, where $\kappa$ and $\lambda$ are the constants from Theorem* 3.1.

*Proof.* Let $uv \in E$ be an edge of $G$. If both endpoints $u, v \notin \bar{V}$, then each endpoint is either a terminal or a node at distance 0 from some terminal; hence $d(f(u), f(v)) = \delta(u,v)$.

If $u \notin \bar{V}$ and $v \in \bar{V}$, then $\delta(f(u), u) = 0$, and $v$, together with the cluster $C$ that contains it, is assigned to some terminal $i$. By the definition of the rounding procedure, there is a node $x \in C$ such that $\delta(x, i) = A_x \le A_v$. Combining Claim 3.3 and Lemma 3.4, we have $\delta(v, x) \le A_v$. Therefore, using the triangle inequality, $\delta(v, i) \le 2A_v$. Using the triangle inequality again, $d(f(u), f(v)) = \delta(f(u), f(v)) \le \delta(f(u), v) + \delta(v, i) \le \delta(f(u), v) + 2A_v$. However, $\delta(u,v) = \delta(f(u), v) \ge A_v$. Therefore, for any $u \notin \bar{V}$ and $v \in \bar{V}$

$$d(f(u), f(v)) \le 3\delta(u,v). \tag{3.1}$$

We are left with the edges $uv \in \bar{E}$. For $u \in \bar{V}$, let $C(u)$ denote the cluster containing $u$. Then

$$\sum_{uv \in \bar{E}} d(f(u), f(v))c(u,v) = \sum_{uv \in \bar{E}:\ C(u) \ne C(v)} d(f(u), f(v))c(u,v).$$

However, $d(f(u), f(v)) = \delta(f(u), f(v)) \le \delta(f(u), u) + \delta(u,v) + \delta(v, f(v)) \le \delta(u,v) + 2A_u + 2A_v \le \delta(u,v) + 4\max\{A_u, A_v\}$. The second inequality follows from the fact that, by the definition of the algorithm, for every nonterminal $u$, $f(u)$ is a terminal closest to some $x \in C(u)$ with $\delta(f(u), x) = A_x \le A_u$. As argued in the previous case, $\delta(x, u) \le A_u$. The inequality follows because, by the triangle inequality, $\delta(f(u), u) \le \delta(f(u), x) + \delta(x, u)$. Therefore,

$$\sum_{uv \in \bar{E}:\ C(u) \ne C(v)} d(f(u), f(v))c(u,v) \le \sum_{uv \in \bar{E}:\ C(u) \ne C(v)} \delta(u,v)c(u,v) + 4 \sum_{uv \in \bar{E}:\ C(u) \ne C(v)} \tilde{c}(uv)$$

$$\leq Z + 4 \sum_{uv \in \bar{E}:\ C(u) \neq C(v)} \tilde{c}(uv).$$

Now Claim 3.3 states that $\sum_{uv \in \bar{E}:\ C(u) \neq C(v)} \tilde{c}(uv) \leq 4\kappa\lambda r^3 Z$, and using this together with (3.1), we finish the proof of Lemma 3.5. ◻

We conclude with the main result of this section.

THEOREM 3.6. *Let $r > 0$ be a fixed integer. There is a deterministic polynomial-time $(4 + 16\kappa\lambda r^3)$-approximation algorithm for* 0-EXTENSION *in $K_{r,r}$-minor–free weighted graphs, where $\kappa$ and $\lambda$ are the constants from Theorem* 3.1.

*Proof.* Solve (MET) optimally and then use the rounding procedure from this section, which clearly can be implemented in polynomial time. Lemma 3.5 establishes the performance guarantee of this algorithm. ◻

**4. The integrality ratio.** In this section we use the max flow-min cut theorem to prove the following lower bound on the integrality ratio of the natural relaxation.

THEOREM 4.1. *There are $c > 0$ and infinitely many positive integers $k$ such that there is an instance of* 0-EXTENSION *with $k$ terminals for which the optimal value of the objective function is at least $c\sqrt{\lg k}$ times the optimal value of the relaxation.*

*Proof.* There are fixed positive $\Delta$ and $\alpha$ such that there is an infinite family of expanders of maximum degree at most $\Delta$ having expansion at least $\alpha$, i.e., graphs $G$ of maximum degree at most $\Delta$ such that, for any subset $S$ of at most $|V(G)|/2$ nodes, there are at least $\alpha|S|$ nodes not in $S$ which are adjacent to at least one node of $S$. For any expander $G$ with $n = |V(G)|$ sufficiently large, define $l = \lceil \sqrt{\lceil \lg n \rceil} \rceil \leq n$ and $k = \lceil n/l \rceil$. Choose any $k$ distinct nodes $h_1, h_2, \ldots, h_k$ in $V$. For $i = 1$ to $k$, add $l$ new nodes and $l$ new edges to the current graph, forming a new path $P_i$ starting at $h_i$ and ending at some new node; label that new node $i$. Let the new graph be $G' = (V', E')$. Now $n' = |V'| = n + kl \leq n + (1 + n/l) \cdot l \leq n + (n + l) \leq n + 2n = 3n$ vertices. $|E'| = |E| + kl \leq n(\Delta/2 + 2)$.

Now define an instance $I$ of 0-EXTENSION as follows. The vertex set is $V'$. The set $T$ of terminals is $\{1, 2, 3, \ldots, k\}$. Define $d(i,j)$, for terminals $1 \leq i, j \leq k$, to be the distance in $G'$ between $i$ and $j$. Define $c(u,v)$ to be 1 if $uv \in E'$, and $c(u,v) = 0$ otherwise.

We now show that the integrality ratio for this instance $I$ is large. First, we study the relaxation. Define $\delta(u,v)$ to be the distance in $G'$ between $u$ and $v$. It is clear that $\delta(i,j) = d(i,j)$ if $i, j$ in $T$. It is also clear that $\delta$ is a semimetric on $G'$. It follows that $\sum_{u<v} c(u,v)\delta(u,v) = |E'| \leq (\Delta/2 + 2)n$ (since adjacent vertices in $G'$ are at distance 1).

Now we prove that there is a universal $c > 0$ such that any feasible solution to $I$, i.e., any function $f : V' \to T$ with $f(t) = t$ for all terminals $t \in T$, satisfies $\sum_{u<v} c(u,v)d(f(u), f(v)) \geq cn\sqrt{\lg n}$. Note first of all that the minimum distance between two distinct terminals $i, j$ is at least $2l \geq 2\sqrt{\lg n}$. We will see in Lemma 4.2, however, that there are at least $k/2$ terminals for which the distance to terminal $i^*$ is at least $\epsilon \lg n$, $\epsilon$ a fixed positive constant, not just $\sqrt{\lg n}$, for any $i^* \in T$.

LEMMA 4.2. *For any $i^* \in T$, there are at least $k/2$ vertices $h_i$ in $G$ whose distance from $h_{i^*}$ exceeds $a = \lceil \frac{\lg k}{2 \lg \Delta} \rceil$.*

*Proof.* For a contradiction, assume that there are more than $k/2$ vertices of $G$ at distance at most $a$ from $h_{i^*}$. By the degree bound, the number of vertices in $G$ at distance at most $a$ from $h_{i^*}$ is at most $1 + \Delta^1 + \Delta^2 + \cdots + \Delta^a < \Delta^{a+1}$. Hence $k/2 < \Delta^{a+1}$. Hence $-1 + \frac{-1 + \lg k}{\lg \Delta} < a$, $\lg k < 2 + 4 \lg \Delta$, and $k < 4\Delta^4$. Require that $n$ be large enough that $k$, which goes to $\infty$ with $n$, is at least $4\Delta^4$, giving us a contradiction. $\square$

Let $R_i = \{v \in V \subseteq V' | f(v) = i\}$. We have two cases.

*Case* 1. $|R_i| \leq n/2$ for all $i$. If $uv \in E$, $u \in R_i$, $v \in R_j$, $i \neq j$, then $d(f(u), f(v)) = d(i,j) = \delta(i,j) \geq 2\sqrt{\lg n}$. Because $G$ is an expander and $|R_i| \leq n/2$ for all $i$, for each $i$ the number of edges $uv$, $u \in R_i$, $v \notin R_i$, is at least $\alpha|R_i|$. Hence $\sum_{u<v,u,v\in V} c(u,v)d(f(u),f(v)) \geq (1/2) \times \sum_{i=1}^{k} \alpha|R_i|(2\sqrt{\lg n}) = \alpha \cdot n\sqrt{\lg n}$. (Each "cross edge" is counted twice.)

*Case* 2. Some $R_i$, say $R_1$, has size exceeding $n/2$. We will use expansion and the max flow-min cut theorem to prove our theorem.

Choose any $\lceil k/2 \rceil$ terminals $i$ such that the distance in $G$ between $h_i$ and $h_1$ is at least $a = \lceil \frac{\lg k}{2 \lg \Delta} \rceil$; by Lemma 4.2, such terminals exist. Let $F$ be the set of chosen terminals. Insist that $n \geq 16$, so that $a \geq \frac{\lg n}{4 \lg \Delta}$.

Let $V^* = V \cup (\cup_{i\in F} P_i)$. Build a (directed) network $N$ on $V^* \cup \{s,t\}$, $s,t$ being new nodes, as follows. Start with the subgraph of $G'$ induced by $V^*$, and replace each edge by a pair of antiparallel arcs, each of capacity one. Add arcs $(u,t)$ for all $u \in R_1$, each of capacity $\infty$. Add arcs $(s,i)$ for all $i \in F$, each of capacity $\infty$. Because $R_1 \subseteq V$ and, for all $i \in F$, $i \notin V$, $N$ has a finite capacity $s \to t$ cut defined by $\{s\} \cup F$.

Now choose any finite capacity $s \to t$ cut $C^* = (S^*, \bar{S^*})$ in $N$, $s \in S^*$, $t \notin S^*$. $S^* \supseteq F$ and $S^* \cap R_1 = \emptyset$. Let $S = S^* \cap V$ (possibly $S = \emptyset$). Because $|R_1| \geq n/2$, $|S| \leq n/2$. Let $C$ be the set of arcs $(u,v)$ with $u \in S, v \in V - S$. By the expansion of $G$, $|C| \geq \alpha|S|$. Let $M = \{i | i \in F, h_i \notin S\}$. Corresponding to each $i \in F$ such that $h_i \notin S$ there is at least one an arc of $P_i$ in $C^* - C$, $|M|$ in total. Thus the total number of arcs in $C^*$ is at least $\alpha|S| + |M| \geq \alpha(|S| + |M|) \geq \alpha|F| \geq \alpha k/2$, the penultimate inequality following from the fact that either $h_i \in S$ or $i \in M$, for all $i \in F$. It follows that the minimum capacity of an $s \to t$ cut in $N$ is at least $\alpha k/2$.

By the max flow-min cut theorem, there are at least $\alpha k/2$ arc-disjoint paths from an $i \in F$ to some vertex in $R_1$. If $Q = \langle i = v_{i0}, v_{i1}, v_{i2}, v_{i3}, \ldots, v_{is} \in R_1 \rangle$ is such a path, then, using $f(i) = i$, $f(v_{is}) = 1$, we have $\sum_{j=0}^{s-1} d(f(v_{ij}), f(v_{i,j+1})) \geq d(f(i), f(v_{is})) = d(i,1) \geq a$, $a$ being at least $\frac{\lg n}{4 \lg \Delta}$. Since the paths are arc-disjoint and there are at least $\frac{\alpha k}{2}$ of them, we infer that $\sum_{u<v:u,v\in V'} c(u,v)d(f(u),f(v)) \geq \frac{\alpha k}{2} \frac{\lg n}{4 \lg \Delta}$.

Using the definition of $k$, this last sum is at least $(\alpha/(16 \lg \Delta)) \cdot n\sqrt{\lg n}$. Since we have a feasible solution to (MET) of value at most $|E'| \leq (\Delta/2 + 2)n$, the ratio between the two is at least

$$\frac{\alpha}{(8 \lg \Delta)(\Delta + 4)} \sqrt{\lg n} \geq \frac{\alpha}{(8 \lg \Delta)(\Delta + 4)} \sqrt{\lg k}.$$

Choose $c = \alpha/((8 \lg \Delta)(\Delta + 4))$, and the proof of Theorem 4.1 is complete. $\square$

The following theorem shows that the above analysis is asymptotically tight. It also suggests an alternative rounding procedure that for some instances performs better than the results in section 2 (though in general it is far worse).

THEOREM 4.3. *There is a polynomial-time algorithm that takes as input a connected graph $G = (V,E)$ and a subset $T \subseteq V$ of terminals and computes a function $f : V \to T$ with $f(i) = i$ for all $i \in T$ such that*

$$\sum_{uv\in E} d(f(u), f(v)) \leq 3\sqrt{d_{\max}}|E|,$$

*where $d(u,v)$ is the minimum number of edges in a path between $u$ and $v$ and $d_{\max} = \max_{uv} d(u,v)$.*

*Proof.* Add a new vertex $s$ to $G$ and connect $s$ to all the terminals. Run a breadth-first search starting at $s$, computing for every node $v \in V$ its distance $l(v)$ from $s$. Note that for every $v \in V$, $1 \le l(v) \le d_{\max} + 1$. Partition $E$ into classes $C_i$, for $i = 1, 2, \ldots, d_{\max}+1$: Place edge $uv \in E$ in $C_i$ for $i = \min\{l(u), l(v)\}$. Let $r$ be the smallest positive integer such that $|C_r| \le |E|/\sqrt{d_{\max}}$. As $\sum_i |C_i| = |E|$, $r \le \sqrt{d_{\max}}$. We now define $f$. Let $t$ be an arbitrary terminal. For every $v \in V$ with $l(v) > r$, set $f(v) = t$. For every $v \in V$ with $l(v) \le r$, set $f(v) = t_v$, where $t_v$ is a terminal closest to $v$ in $G$.

If $uv \in C_i$, then $d(u, t_u) \le i$ and $d(v, t_v) \le i$, and at least one of the inequalities is strict. Therefore $d(t_u, t_v) \le d(t_u, u) + 1 + d(v, t_v) \le 2i$. Consider an edge $uv \in C_i$. If $i \le r - 1$, then both $l(u), l(v) \le r$, so $d(f(u), f(v)) = d(t_u, t_v) \le 2i \le 2r - 2$. If $i > r$, then both $l(u), l(v) > r$, so $d(f(u), f(v)) = d(t, t) = 0$. For the remaining case of $i = r$ we use the trivial bound $d(f(u), f(v)) \le d_{\max}$. Using the bounds on $r$ and on $|C_r|$, we have

$$\sum_{uv\in E} d(f(u), f(v)) \le \sum_{i<r} \sum_{uv\in C_i} (2r-2) + \sum_{uv\in C_r} d_{\max} + \sum_{i>r} \sum_{uv\in C_i} 0 < 3\sqrt{d_{\max}}\ |E|. \qquad \square$$

It is interesting to note that for bounded degree expanders the bounds are much better. Using arguments similar to those of the proofs of Theorems 4.1 and 4.3, we can prove the following result.

THEOREM 4.4.

1. *There are a positive integer $\Delta$ and a constant $\kappa > 0$ such that for infinitely many $k$ there is an expander $G = (V, E)$ with maximum degree at most $\Delta$, $|V|$ being $O(k \log k / \log\log k)$, and a set $T \subseteq V$ of size $k$, such that the integrality ratio of* (MET) *on the* 0*-extension instance defined by $G$, $T$, and the $G$-path metric on $T$ is at least $\kappa \log\log k$.*
2. *For all positive constants $\Delta$ and $\alpha$, there is a $\lambda$ such that if $n$ is sufficiently large, the optimal cost of* 0-EXTENSION *on the* 0*-extension instance defined by any $n$-node expander $G$ of maximum degree at most $\Delta$, with expansion constant at least $\alpha$, any set $T \subseteq V$ of terminals, and the $G$-path metric on $T$, is at most $\lambda n \lg\lg n$ (and there is a polynomial-time algorithm which computes a solution to the* 0*-extension problem of cost at most $\lambda n \lg\lg n$).*

*Proof sketch.* For the first part, choose a family of expanders of maximum degree at most $\Delta$. Given $k$, choose an $n$ and an $n$-node expander from the family such that $k$ is approximately equal to $n(\lg\lg n)/\lg n$. Now we modify the proof of Theorem 4.1. Instead of choosing $h_1, h_2, \ldots, h_k$ to be any $k$ nodes, choose $k$ nodes with minimum pairwise distance $\Omega(\lg\lg n)$, as follows. Choose the first node arbitrarily and choose the $j$th node to be at distance $\Omega(\lg\lg n)$ from the $j - 1$ previously chosen nodes. The iterative choices are possible since, for some suitable constant $c$, the number of nodes within distance at most $c \lg\lg n$ from $j - 1$ given points is at most $j\Delta^{1+c\lg\lg n} < n$. Call the $k$ nodes $1, 2, \ldots, k$, and make them the terminals. The 0-extension instance is now defined on this graph $G$, relative to its shortest path metric. It is clear that the optimal value of (MET) is $O(n)$.

Given any vertex $v$ of $G$, arguing as in the proof of Lemma 4.2, there are at least $k/2$ terminals in $G$ whose distance from $v$ is at least $a$, with $a$ being $\Omega(\log n)$.

Now we study Cases 1 and 2 from the proof of Theorem 4.1. In Case 1, we have $d(f(u), f(v)) = d(i, j)$, which is $\Omega(\log\log n)$, so the total cost is $\Omega(n \log\log n)$.

The argument of Case 2 applies as before: There are at least $\alpha k/2$ (which is $\Omega(n(\log\log n)/\log n)$) paths, each contributing at least $a$ (which is $\Omega(\log n)$), or $\Omega(n \log\log n)$ in total.

For the second part of Theorem 4.4, let $\Delta, \alpha$ be positive and let $G = (V, E)$ be an $n$-node expander of maximum degree at most $\Delta$ and expansion constant at least $\alpha$. Let $T = \{1, 2, \ldots, k\} \subseteq V$ be the set of terminals. Note that there is a constant $C = C(\Delta, \alpha)$ such that $C \lg n$ bounds the diameter from above.

Consider the 0-extension instance defined by $G$, $T$, and the $G$-path metric on $T$. There are two cases.

*Case* 1. If $k \leq n/\lg n$, set $f(v) = 1$ for all $v \in V - T$. The number of edges $\{u, v\}$ that are cut is at most $\Delta \cdot k$, and for each, $d(f(u), f(v)) \leq C \lg n$. Hence the total cost is at most $C\Delta k \lg n \leq C\Delta n$.

*Case* 2. If $k > n/\lg n$, add a dummy source $s$ which is adjacent to all (and only) the terminals. Do a breadth-first search starting from $s$ until there are no more than $n/\lg n$ unreached vertices. Since $\Delta, \alpha$ are constant, the number of reached nodes increases by a constant factor in each BFS step, until the number of reached nodes exceeds $n/2$. Afterward, the number of unreached nodes drops by a constant factor in each BFS step. Altogether, where $d$ is the number of BFS steps needed, $d$ is $O(\lg \lg n)$ (because $\Delta, \alpha$ are constant). Now assign $f(v)$ for nonterminals $v$ as follows. If $v$ is unreached, set $f(v) = 1$. Otherwise, set $f(v)$ equal to the terminal nearest to $v$. Now consider $\sum c(u, v)d(f(u), f(v))$. If $u, v$ are both reached, the contribution $c(u, v)d(f(u), f(v)) \leq d + 1 + d$, which is $O(\lg \lg n)$. If $u, v$ are both unreached, the contribution is 0. There are at most $\Delta n/\lg n$ edges $\{u, v\}$ with $u$ reached and $v$ not, and each contributes at most $C \lg n$, or at most $\Delta C n$ in total for these edges. Hence the overall total is $O(n \lg \lg n)$. $\square$

**Acknowledgment.** The authors gratefully acknowledge the careful readings by and helpful suggestions from the referees.

## REFERENCES

[1] Y. Bartal, *On approximating arbitrary metrics by tree metrics*, in Proceedings of the 30th Annual ACM Symposium on Theory of Computing, Dallas, TX, 1998, pp. 161–168.

[2] G. Calinescu, H. Karloff, and Y. Rabani, *An improved approximation algorithm for multiway cut*, J. Comput. Syst. Sci., 60 (2000), pp. 564–574.

[3] C. Chekuri, S. Khanna, J. Naor, and L. Zosin, *Approximation algorithms for the metric labeling problem via a new linear programming formulation*, in Proceedings of the 12th Annual ACM-SIAM Symposium on Discrete Algorithms, Washington, DC, 2001, SIAM, Philadelphia, pp. 109–118.

[4] J. Chuzhoy, *Private communication*, 1999.

[5] W. H. Cunningham, *The optimal multiterminal cut problem*, in DIMACS Ser. Discrete Math. Theoret. Comput. Sci. 5, AMS, Providence, RI, 1991, pp. 105–120.

[6] W. H. Cunningham and L. Tang, *Optimal* 3*-terminal cuts and linear programming*, in Proceedings of the 7th Conference on Integer Programming and Combinational Optimization, Graz, Austria, 1999, pp. 114–125.

[7] E. Dahlhaus, D. S. Johnson, C. H. Papadimitriou, P. D. Seymour, and M. Yannakakis, *The Complexity of Multiway Cuts*, extended abstract, 1983.

[8] E. Dahlhaus, D. S. Johnson, C. H. Papadimitriou, P. D. Seymour, and M. Yannakakis, *The Complexity of Multiterminal Cuts*, SIAM J. Comput., 23 (1994), pp. 864–894.

[9] N. Garg, V. V. Vazirani, and M. Yannakakis, *Primal-dual approximation algorithms for integral flow and multicut in trees*, Algorithmica, 18 (1997), pp. 3–20.

[10] N. Garg, V. V. Vazirani, and M. Yannakakis, *Approximate max-flow min-(multi)cut theorems and their applications*, SIAM J. Comput., 25 (1996), pp. 235–251.

[11] A. Gupta and É. Tardos, *A constant factor approximation algorithm for a class of classification problems*, in Proceedings of the 32nd Annual ACM Symposium on Theory of Computing, Portland, OR, 2000, pp. 652–658.

[12] D. R. Karger, P. Klein, C. Stein, M. Thorup, and N. E. Young, *Rounding algorithms for a geometric embedding of minimum multiway cut*, Math. Oper. Res., 29 (2004), pp. 436–461.

[13] A. V. Karzanov, *Minimum* 0*-extension of graph metrics*, European J. Combin., 19 (1998), pp. 71–101.

[14] P. Klein, A. Agarwal, R. Ravi, and S. Rao, *Approximation through multicommodity flow*, in Proceedings of the 31st IEEE Symposium on Foundations of Computer Science, St. Louis, MO, 1990, pp. 726–737.

[15] P. Klein, S. A. Plotkin, and S. Rao, *Excluded minors, network decomposition, and multicommodity flows*, in Proceedings of the 25th Annual ACM Symposium on Theory of Computing, San Diego, CA, 1993, pp. 682–690.

[16] J. Kleinberg and É. Tardos, *Approximation algorithms for classification problems with pairwise relationships: Metric labeling and Markov random fields*, in Proceedings of the 40th Annual IEEE Symposium on Foundations of Computer Science, New York, 1999, pp. 14–23.

[17] F. T. Leighton and S. Rao, *An approximate max-flow min-cut theorem for uniform multicommodity flow problems with applications to approximation algorithms*, J. ACM, 46 (1998), pp. 787–832.

[18] É. Tardos and V. V. Vazirani, *Improved bounds for the max-flow min-multicut ratio for planar and $K_{r,r}$-free graphs*, Inform. Process Lett., 47 (1993), pp. 77–80.

SIAM J. Comput.
Vol. 34, No. 2, pp. 373–387

# AN OPTIMAL ALGORITHM FOR THE MAXIMUM-DENSITY SEGMENT PROBLEM*

KAI-MIN CHUNG† AND HSUEH-I LU‡

**Abstract.** We address a fundamental problem arising from analysis of biomolecular sequences. The input consists of two numbers $w_{\min}$ and $w_{\max}$ and a sequence $S$ of $n$ number pairs $(a_i, w_i)$ with $w_i > 0$. Let *segment* $S(i,j)$ of $S$ be the consecutive subsequence of $S$ between indices $i$ and $j$. The *density* of $S(i,j)$ is $d(i,j) = (a_i + a_{i+1} + \cdots + a_j)/(w_i + w_{i+1} + \cdots + w_j)$. The *maximum-density segment problem* is to find a maximum-density segment over all segments $S(i,j)$ with $w_{\min} \le w_i + w_{i+1} + \cdots + w_j \le w_{\max}$. The best previously known algorithm for the problem, due to Goldwasser, Kao, and Lu [*Proceedings of the Second International Workshop on Algorithms in Bioinformatics*, R. Guigó and D. Gusfield, eds., Lecture Notes in Comput. Sci. 2452, Springer-Verlag, New York, 2002, pp. 157–171], runs in $O(n \log(w_{\max} - w_{\min} + 1))$ time. In the present paper, we solve the problem in $O(n)$ time. Our approach bypasses the complicated *right-skew decomposition*, introduced by Lin, Jiang, and Chao [*J. Comput. System Sci.*, 65 (2002), pp. 570–586]. As a result, our algorithm has the capability to process the input sequence in an online manner, which is an important feature for dealing with genome-scale sequences. Moreover, for a type of input sequences $S$ representable in $O(m)$ space, we show how to exploit the sparsity of $S$ and solve the maximum-density segment problem for $S$ in $O(m)$ time.

**Key words.** bioinformatics, biological sequence analysis, maximum-density segment, slope selection, computational geometry, sequence algorithm, data structure

**AMS subject classifications.** 68P05, 68Q25, 68R05, 68U05, 68W05, 68W40, 92-08, 92D20

**DOI.** 10.1137/S0097539704440430

## 1. Introduction.

We address the following fundamental problem: The input consists of two numbers $w_{\min}$ and $w_{\max}$ and a sequence $S$ of number pairs $(a_i, w_i)$ with $w_i > 0$ for $i = 1, \ldots, n$. A *segment* $S(i,j)$ is a consecutive subsequence of $S$ from position $i$ to position $j$. The *width* $w(i,j)$ of $S(i,j)$ is $w_i + w_{i+1} + \cdots + w_j$. The *density* $d(i,j)$ of $S(i,j)$ is $(a_i + a_{i+1} + \cdots + a_j)/w(i,j)$. It is not difficult to see that with an $O(n)$-time preprocessing to compute all $O(n)$ prefix sums $a_1 + a_2 + \cdots + a_j$ and $w_1 + w_2 + \cdots + w_j$, the density of any segment can be computed in $O(1)$ time. $S(i,j)$ is *feasible* if $w_{\min} \le w(i,j) \le w_{\max}$. The *maximum-density segment* problem is to find a maximum-density segment over all $O(n(w_{\max} - w_{\min} + 1))$ feasible segments.

This problem arises from the investigation of the nonuniformity of nucleotide composition within genomic sequences, which was first revealed through thermal melting and gradient centrifugation experiments [21, 28]. The GC content of the DNA sequences in all organisms varies from 25% to 75%. GC-ratios have the greatest variation among bacterial DNA sequences, while the typical GC-ratios of mammalian

*Received by the editors February 3, 2004; accepted for publication (in revised form) July 7, 2004; published electronically December 1, 2004. A preliminary version of this paper appeared in *Proceedings of the* 11th *Annual European Symposium on Algorithms* (Budapest, Hungary, 2003), G. Di Battista and U. Zwick, eds., Lecture Notes in Comput. Sci. 2832, Springer-Verlag, New York, 2003, pp. 136–147.

http://www.siam.org/journals/sicomp/34-2/44043.html

†Institute of Information Science, Academia Sinica. Part of this work was done while this author was an undergraduate student in the Department of Computer Science and Information Engineering, National Taiwan University.

‡Corresponding author. Institute of Information Science, Academia Sinica, 128 Academia Road, Section 2, Taipei 115, Taiwan, Republic of China (hil@iis.sinica.edu.tw, www.iis.sinica.edu.tw/∼hil/). This author's research was supported in part by NSC grants 91-2215-E-001-001 and 92-2218-E-001-001.

genomes stay in the range 45–50%. Despite intensive research effort in the past two decades, the underlying causes of the observed heterogeneity remain debatable [2, 3, 5, 8, 9, 10, 11, 18, 40, 42]. Researchers [32, 39] observed that the extent of the compositional heterogeneity in a genomic sequence strongly correlates with its GC content regardless of genome size. Other investigations showed that gene length [7], gene density [44], patterns of codon usage [37], distribution of different classes of repetitive elements [7, 38], number of isochores [2], lengths of isochores [32], and recombination rate within chromosomes [12] are all correlated with GC content. More research related to GC-rich segments can be found in [16, 17, 20, 23, 29, 31, 35, 41, 43] and references therein.

In the most basic form of the maximum-density segment problem, the sequence $S$ corresponds to the given DNA sequence, where $a_i = 1$ if the corresponding nucleotide in the DNA sequence is a G or C, and $a_i = 0$ otherwise. In the work of Huang [19], sequence entries took on values of $p$ and $1 - p$ for some real number $0 \le p \le 1$. More generally, we can look for regions where a given set of patterns occurs very often. In such applications, $a_i$ could be the relative frequency with which the corresponding DNA segments appear in the given patterns. Further natural applications of this problem can be designed for sophisticated sequence analysis such as mismatch density [36], ungapped local alignments [1], annotated multiple sequence alignments [39], promoter mapping [22], and promoter recognition [33].

For the *uniform* case, i.e., $w_i = 1$ for all indices $i$, Nekrutenko and Li [32] and Rice, Longden, and Bleasby [34] employed algorithms for the case $w_{\min} = w_{\max}$, which is trivially solvable in $O(n)$ time. More generally, when $w_{\min} \ne w_{\max}$, the problem is also easily solvable in $O(n(w_{\max} - w_{\min} + 1))$ time, linear in the number of feasible segments. Huang [19] studied the case where $w_{\max} = n$, i.e., there is effectively no upper bound on the width of the desired maximum-density segments. He observed that an optimal segment exists with width at most $2w_{\min} - 1$. Therefore, this case is equivalent to the case with $w_{\max} = 2w_{\min} - 1$ and can be solved in $O(nw_{\min})$ time in a straightforward manner. Lin, Jiang, and Chao [27] gave an $O(n \log w_{\min})$-time algorithm for this case based on right-skew decompositions of a sequence. (See [26] for related software.) The case with general $w_{\max}$ was first investigated by Goldwasser, Kao, and Lu [13, 14], who gave an $O(n)$-time algorithm for the uniform case. Recently, Kim [25] showed an alternative algorithm based upon a geometric interpretation of the problem, which basically relates the maximum-density segment problem to the fundamental *slope selection problem* in computational geometry [4, 6, 24, 30]. Unfortunately, Kim's analysis of time complexity has a flaw which seems difficult to fix.[1] For the general (i.e., *nonuniform*) case, Goldwasser, Kao, and Lu [13] also gave an $O(n \log(w_{\max} - w_{\min} + 1))$-time algorithm. By bypassing the complicated preprocessing step required in [13], we successfully reduce the time required for the general case down to $O(n)$. Our result is based upon the following set of equations, stating that the order of $d(x, y)$, $d(y + 1, z)$, and $d(x, z)$ with $x \le y < z$ can be determined by the

[1]Kim claims that all the progressive updates of the lower convex hulls $L_j \cup R_j$ can be done in overall linear time. The paper only sketches how to obtain $L_{j+1} \cup R_{j+1}$ from $L_j \cup R_j$. (See the fourth-to-last paragraph of p. 340 in [25].) Unfortunately, Kim seems to overlook the marginal cases when the upper bound $w_{\max}$ forces the $p_z$ of $L_j \cup R_j$ to be deleted from $L_{j+1} \cup R_{j+1}$. As a result, obtaining $L_{j+1} \cup R_{j+1}$ from $L_j \cup R_j$ could be much more complicated than Kim's sketch. A naive implementation of Kim's algorithm still takes $\Omega(n(w_{\max} - w_{\min} + 1))$ time in the worst case. We believe that any correct implementation of Kim's algorithm requires $\Omega(n \log(w_{\max} - w_{\min} + 1))$ time in the worse case.

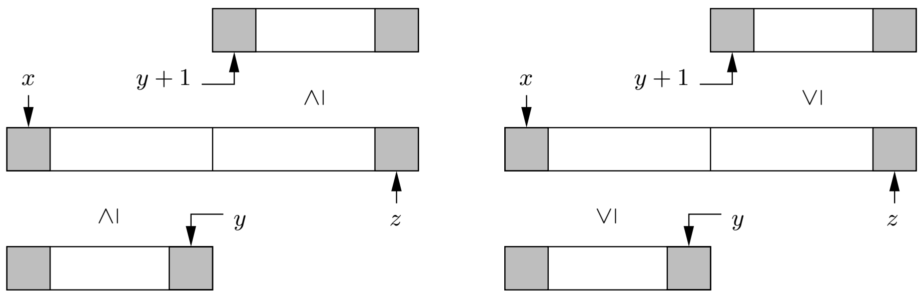


FIG. 1.1. *An illustration for* (1.1)*: There exist only two possibilities for the order among* $d(x,y), d(x,z), d(y+1,z)$.

order of any two of them:

$$\begin{aligned} d(x,y) \le d(y+1,z) \Leftrightarrow d(x,y) \le d(x,z) \Leftrightarrow d(x,z) \le d(y+1,z),\\ d(x,y) \ge d(y+1,z) \Leftrightarrow d(x,y) \ge d(x,z) \Leftrightarrow d(x,z) \ge d(y+1,z). \end{aligned} \tag{1.1}$$

(Both equations can be easily verified by observing the existence of some number $\rho$ with $0 < \rho < 1$ and $d(x,z) = \rho \cdot d(x,y) + (1-\rho) \cdot d(y+1,z)$. See Figure 1.1.) Our algorithm is capable of processing the input sequence in an online manner, which is an important feature for dealing with genome-scale sequences.

For bioinformatics applications, e.g., in [1, 22, 33, 36, 39], the input sequence $S$ is usually very *sparse*. That is, $S$ can be represented by $m = o(n)$ triples

$$(a'_1, w'_1, n_1), (a'_2, w'_2, n_2), \ldots, (a'_m, w'_m, n_m)$$

with $0 = n_0 < n_1 < n_2 < \cdots < n_m = n$ to signify that $(a_i, w_i) = (a'_j, w'_j)$ holds for all indices $i$ and $j$ with $n_{j-1} < i \le n_j$ and $1 \le j \le m$. If $w'_j = 1$ holds for all $1 \le j \le m$, we show how to exploit the sparsity of $S$ and solve the maximum-density problem for $S$ given in the above compact representation in $O(m)$ time.

The remainder of the paper is organized as follows. Section 2 shows the main algorithm. Section 3 explains how to cope with the simple case that the width upper bound $w_{\max}$ is ineffective. Section 4 takes care of the more complicated case that $w_{\max}$ is effective. Section 5 explains how to exploit the sparsity of the input sequence for the uniform case.

**2. The main algorithm.** For any integers $x$ and $y$, let $[x, y]$ denote the set $\{x, x+1, \ldots, y\}$. Without loss of generality, we may assume that $w_1 + w_2 + \cdots + w_n \ge w_{\min}$ and $w_i \le w_{\max}$ holds for each $i = 1, 2, \ldots, n$. Throughout the paper, we need the following definitions and notation with respect to the input length-$n$ sequence $S$ and width bounds $w_{\min}$ and $w_{\max}$. Define $\phi(x, y)$ to be the largest index $z \in [x, y]$ that minimizes $d(x, z)$. That is, $S(x, \phi(x, y))$ is the longest minimum-density prefix of $S(x, y)$. Let $j_0$ be the smallest index with $w(1, j_0) \ge w_{\min}$. Let $J = [j_0, n]$. For each $j \in J$, let $\ell_j$ be the smallest index $i$ with $w(i, j) \le w_{\max}$, and let $r_j$ be the largest index $i$ with $w(i, j) \ge w_{\min}$. That is, $S(i, j)$ is feasible if and only if $i \in [\ell_j, r_j]$. (Figure 2.1 is an illustration for the definitions of $\ell_j$ and $r_j$.) Clearly, for the uniform case, we have $\ell_{i+1} = \ell_i + 1$ and $r_{i+1} = r_i + 1$. As for the general case, we know only that $\ell_j$ and $r_j$ are both (not necessarily strictly) increasing. One can easily compute all $\ell_j$ and $r_j$ in $O(n)$ time.

Let $i^*_j$ be the largest index $k \in [\ell_j, r_j]$ with $d(k, j) = \max\{d(i, j) \mid i \in [\ell_j, r_j]\}$. There must be an index $j^*$ such that $S(i^*_{j^*}, j^*)$ is a maximum-density segment of

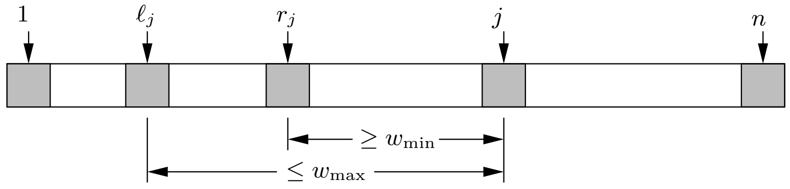


FIG. 2.1. *An illustration for the definitions of $\ell_j$ and $r_j$.*

```
ALGORITHM MAIN.
1    let $i_{j_0-1} = 1$;
2    **for** $j = j_0$ **to** $n$ **do** {
3        let $i_j = $ BEST$(\max(i_{j-1}, \ell_j), r_j, j)$;
4        **output** $(i_j, j)$;
5    }

FUNCTION BEST$(\ell, r, j)$.
1    let $i = \ell$;
2    **while** $i < r$ **and** $d(i, \phi(i, r-1)) \leq d(i, j)$ **do**
3        let $i = \phi(i, r-1) + 1$;
4    **return** $i$;
```

FIG. 2.2. *Our main algorithm.*

$S$. Therefore, a natural but seemingly difficult possibility for solving the maximum-density segment problem would be to compute $i_j^*$ for all indices $j \in J$ in $O(n)$ time. Instead, our approach is to compute a pair $(i_j, j)$ of indices with $i_j \in [\ell_j, r_j]$ for each index $j \in J$ by the algorithm shown in Figure 2.2. More specifically, each iteration of our algorithm, based upon (1.1), keeps chopping off the lowest-density prefix without affecting the feasibility of the remaining segment until its density does not go any higher. For brevity of presentation, each algorithm throughout the paper outputs a linear number of index pairs $(i, j)$. (See, e.g., step 4 of algorithm MAIN shown in Figure 2.2.) Clearly, it takes a linear-time postprocessing for the produced index pairs to obtain one that maximizes $d(i, j)$. The rest of the section ensures the correctness of our algorithm by showing $i_{j^*} = i_{j^*}^*$, and thus reduces the maximum-density segment problem to implementing our algorithm to run in $O(n)$ time.

LEMMA 2.1. *The index returned by function call* BEST$(\ell, r, j)$ *is the largest index* $i \in [\ell, r]$ *that maximizes* $d(i, j)$.

*Proof.* Let $i^*$ be the largest index in $[\ell, r]$ that maximizes $d(i, j)$, i.e., $d(i^*, j) = \max_{i \in [\ell, r]} d(i, j)$. Let $i_j$ be the index returned by function call BEST$(\ell, r, j)$. We show $i_j = i^*$ as follows. If $i_j < i^*$, then $i_j < r$. By the condition of the while-loop at step 2 of BEST, we know $d(i_j, \phi(i_j, r-1)) > d(i_j, j)$. By $d(i_j, j) \leq d(i^*, j)$ and (1.1), we have $d(i_j, i^*-1) \leq d(i_j, j)$. It follows that $d(i_j, i^*-1) < d(i_j, \phi(i_j, r-1))$, contradicting the definition of $\phi(i_j, r-1)$.

On the other hand, suppose that $i_j > i^*$. By definition of BEST, there must be an index $i \in [\ell, r]$ with $i < r$, $d(i, \phi(i, r-1)) \leq d(i, j)$, and $i \leq i^* < \phi(i, r-1) + 1$. If $i = i^*$, by (1.1) we have $d(i^*, \phi(i^*, r-1)) \leq d(i^*, j) \leq d(\phi(i^*, r-1) + 1, j)$, where the last inequality contradicts the definition of $i^*$. Now that $i < i^*$, we have $d(i^*, j) \geq d(i, j) \geq d(i, i^*-1) \geq d(i, \phi(i, r-1)) \geq d(i^*, \phi(i, r-1))$, where (a) the first inequality is by definition of $i^*$, (b) the second inequality is by (1.1) and the first inequality, (c) the third inequality is by $i^* \le \phi(i, r-1)$ and the definition of $\phi(i, r-1)$, and (d) the last inequality is by (1.1) and the third inequality. It follows from $d(i^*, j) \ge d(i^*, \phi(i, r-1))$ and (1.1) that $d(\phi(i, r-1)+1, j) \ge d(i^*, j)$, contradicting the definition of $i^*$ by $i^* < \phi(i, r-1)+1$. ∎

THEOREM 2.2. *Algorithm* MAIN *correctly solves the maximum-density segment problem.*

*Proof.* We prove the theorem by showing $i_{j^*} = i^*_{j^*}$. By $\ell_{j_0} = i_{j_0-1} = 1$ and Lemma 2.1, the equality holds if $j^* = j_0$. The rest of the proof assumes $j^* > j_0$. By Lemma 2.1 and $\ell_{j^*} \le i^*_{j^*}$, it suffices to ensure $i_{j^*-1} \le i^*_{j^*}$. Assume for contradiction that there is an index $j \in [j_0, j^*-1]$ with $i_{j-1} \le i^*_{j^*} < i_j$. By $j < j^*$, we know $\ell_j \le i^*_{j^*}$. By Lemma 2.1 and $\max(\ell_j, i_{j-1}) \le i^*_{j^*} < i_j \le r_j$, we have $d(i_j, j) \ge d(i^*_{j^*}, j)$. It follows from (1.1) and $i^*_{j^*} < i_j$ that $d(i^*_{j^*}, j) \ge d(i^*_{j^*}, i_j - 1)$. By $\ell_{j^*} \le i^*_{j^*} < i_j \le r_{j^*}$ and the definition of $j^*$, we know $d(i^*_{j^*}, j^*) > d(i_j, j^*)$. It follows from $i^*_{j^*} < i_j$ and (1.1) that $d(i^*_{j^*}, i_j - 1) > d(i^*_{j^*}, j^*)$. Therefore, $d(i_j, j) \ge d(i^*_{j^*}, j) \ge d(i^*_{j^*}, i_j - 1) > d(i^*_{j^*}, j^*)$, contradicting the definition of $j^*$. ∎

One can verify that the value of $i$ increases by at least 1 each time step 3 of BEST is executed. Therefore, to implement the algorithm to run in $O(n)$ time, it suffices to maintain a data structure to support an $O(1)$-time query for each $\phi(i, r_j - 1)$ in step 2 of BEST.

## 3. Coping with ineffective width upper bound.

When $w_{\max}$ is ineffective, i.e., $w_{\max} \ge w(1, n)$, we have $\ell_j = 1$ for all $j \in J$. Therefore, the function call in step 3 of MAIN is exactly BEST$(i_{j-1}, r_j, j)$. Moreover, during the execution of the function call BEST$(i_{j-1}, r_j, j)$, the value of $i$ can only be $i_{j-1}$, $\phi(i_{j-1}, r_j - 1) + 1$, $\phi(\phi(i_{j-1}, r_j - 1) + 1, r_j - 1) + 1, \ldots, r_j$. Suppose that a subroutine call to UPDATE$(j)$ yields an array $\Phi$ of indices and two indices $p$ and $q$ of $\Phi$ with $p \le q$ and $\Phi[p] = i_{j-1}$ such that the following condition holds.

*Condition $C_j$* : $\Phi[q] = r_j$ and $\Phi[t] = \phi(\Phi[t-1], r_j - 1) + 1$ holds for each index $t \in [p+1, q]$. (See Figure 3.1 for an illustration.)

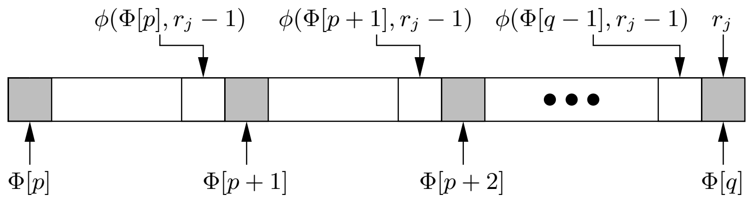


FIG. 3.1. *An illustration for Condition $C_j$.*

Then, the subroutine call to BEST$(i_{j-1}, r_j, j)$ can clearly be replaced by LBEST$(j)$, as defined in Figure 3.2. That is, LBEST$(j)$ can access the value of each $\phi(i, r_j - 1)$ by looking up $\Phi$ in $O(1)$ time. It remains to show how to implement UPDATE$(j)$ such that all $O(n)$ subroutine calls to UPDATE from step 3 of LMAIN run in overall $O(n)$ time. With the initialization of letting $p = q = r_{j_0-1} = \Phi[1] = 1$, as described at step 1 of algorithm LMAIN, Condition $C_{j_0-1}$ clearly holds before the subroutine call to UPDATE$(j_0)$. The following lemma is crucial in ensuring the correctness and efficiency of our implementation shown in Figure 3.2.

LEMMA 3.1. *For each index $j \in J$, the following statements hold:*

1. *If Condition $C_{j-1}$ holds right before calling* UPDATE$(j)$*, then Condition $C_j$ holds right after the subroutine call.*

```
ALGORITHM LMAIN.
1   let $p = q = r_{j_0-1} = \Phi[1] = 1$;
2   for $j = j_0$ to $n$ do {
3       call UPDATE($j$);
4       let $i_j =$ LBEST($j$);
5       output $(i_j, j)$;
6   }

FUNCTION LBEST($j$).
1   while $p < q$ and $d(\Phi[p], \Phi[p+1]-1) \le d(\Phi[p], j)$ do
2       let $p = p + 1$;
3   return $\Phi[p]$;

SUBROUTINE UPDATE($j$).
1   for $r = r_{j-1} + 1$ to $r_j$ do {
2       while $p < q$ and $d(\Phi[q-1], \Phi[q]-1) \ge d(\Phi[q-1], r-1)$ do
3           let $q = q - 1$;
4       let $q = q + 1$;
5       let $\Phi[q] = r$;
6   }
```

FIG. 3.2. *An efficient implementation for the case that $w_{\max}$ is ineffective.*

2. *If Condition $C_j$ holds right before calling* LBEST($j$)*, then the index returned by the function call is exactly that returned by* BEST($\Phi[p], \Phi[q], j$)*.*

*Proof.* Statement 1. We need the following condition.

*Condition $D_r$* : $\Phi[q] = r$ and $\Phi[t] = \phi(\Phi[t-1], r-1) + 1$ holds for each index $t \in [p+1, q]$.

Clearly, Condition $C_j$ is exactly Condition $D_{r_j}$. To prove the statement, we show that if Condition $D_{r-1}$ holds, then Condition $D_r$ holds right after executing steps 2–5 of UPDATE (i.e., an iteration of the for-loop of UPDATE), although the value of $q$ may change.

Consider the moment when step 4 of UPDATE is about to be executed. We first show that if $p < q$, then $\Phi[t] = \phi(\Phi[t-1], r-1) + 1$ holds for each $t \in [p+1, q]$. By the definition of $\phi$, we have

$$\phi(\ell, r-1) = \begin{cases} r-1 & \text{if } d(\ell, \phi(\ell, r-2)) \ge d(\ell, r-1), \\ \phi(\ell, r-2) & \text{otherwise.} \end{cases} \tag{3.1}$$

By Condition $D_{r-1}$, we know $\phi(\Phi[q-1], r-2) = \Phi[q]-1$. With $\ell = \Phi[q-1]$ plugged into (3.1), we know that if $d(\Phi[q-1], \Phi[q]-1) < d(\Phi[q-1], r-1)$, then $\phi(\Phi[q-1], r-1) = \phi(\Phi[q-1], r-2)$. It follows from the condition of step 2 of UPDATE that $\phi(\Phi[q-1], r-1) = \phi(\Phi[q-1], r-2)$. Furthermore, if $\phi(\ell, r-2) < r-2$, then one can prove as follows that $\phi(\phi(\ell, r-2)+1, r-1) = \phi(\phi(\ell, r-2)+1, r-2)$ implies $\phi(\ell, r-1) = \phi(\ell, r-2)$.

Let $m = \phi(\ell, r-2)$. By $\phi(m+1, r-1) = \phi(m+1, r-2)$, we have $d(m+1, \phi(m+1, r-1)) < d(m+1, r-1)$. By definition of $\phi$ and (1.1), we have $d(\ell, m) < d(\ell, \phi(m+1, r-1)) < d(m+1, \phi(m+1, r-1))$. As a result, we have $d(\ell, m) < d(m+1, r-1)$, which by (1.1) implies $d(\ell, m) < d(\ell, r-1)$. Thus $\phi(\ell, r-1) = \phi(\ell, r-2)$.

Therefore, $\phi(\Phi[t], r-1) = \phi(\Phi[t], r-2)$ implies $\phi(\Phi[t-1], r-1) = \phi(\Phi[t-1], r-2)$. As a result, $\phi(\Phi[t], r-1) = \phi(\Phi[t], r-2)$ holds for each $t \in [p, q-1]$. By Condition $D_{r-1}$, we know $\Phi[t] = \phi(\Phi[t-1], r-1) + 1$ holds for each $t \in [p+1, q]$.

Since the value of $q$ will be incremented by 1 after executing step 4 of UPDATE and $\Phi[q+1]$ will equal $r$ after executing step 5 of UPDATE, it remains to show $\phi(\Phi[q], r-1) = r-1$. Clearly, the equality holds if $\Phi[q] = r-1$, i.e., step 3 was not executed. If step 3 was executed at least once, then we know $d(\Phi[q], \Phi[q+1]-1) \geq d(\Phi[q], r-1)$, i.e., $d(\Phi[q], \phi(\Phi[q], r-2)) \geq d(\Phi[q], r-1)$. By plugging $\ell = \Phi[q]$ into (3.1), we know $\phi(\Phi[q], r-1) = r-1$.

Statement 2. By Condition $C_j$, one can easily verify that LBEST$(j)$ is a faithful implementation of BEST$(\Phi[p], \Phi[q], j)$. Therefore, the statement holds. ∎

LEMMA 3.2. *The implementation* LMAIN *solves the maximum-density problem for the case with ineffective* $w_{\max}$ *in* $O(n)$ *time.*

*Proof.* By Lemma 3.1(1) and the definitions of UPDATE and LBEST, both Condition $C_j$ and $\Phi[p] = i_{j-1}$ hold right after the subroutine call to UPDATE$(j)$. By Condition $C_j$ and Lemma 3.1(2), LBEST$(j)$ is a faithful implementation of BEST$(\Phi[p], \Phi[q], j)$. Therefore, the correctness of LMAIN follows from $\Phi[p] = i_{j-1}$, $\Phi[q] = r_j$, and Theorem 2.2.

As for the efficiency of LMAIN, observe that $q - p \geq 0$ holds throughout the execution of LMAIN. Note that each iteration of the while-loops of LBEST and UPDATE decreases the value of $q - p$ by one. Since step 4 of UPDATE, which is the only place that increases the value of $q - p$, increases the value of $q - p$ by one for $O(n)$ times, the overall running time of LMAIN is $O(n)$. ∎

## 4. Coping with effective width upper bound.

In contrast to the previous simple case, when $w_{\max}$ is arbitrary, $\ell_j$ may not always be 1. Therefore, the first argument of the function call in step 3 of MAIN could be $\ell_j$ with $\ell_j > i_{j-1}$. It seems quite difficult to update the corresponding data structure $\Phi$ in overall linear time such that both $\Phi[p] = \max(i_{j-1}, \ell_j)$ and Condition $C_j$ hold throughout the execution of our algorithm. To overcome this difficulty, our algorithm sticks with Condition $C_j$ but allows $\Phi[p] > \max(i_{j-1}, \ell_j)$. As a result, $\max_{j\in J} d(i_j, j)$ may be less than $\max_{j\in J} d(i_j^*, j)$. Fortunately, this potential problem can be resolved if we simultaneously solve a series of variant versions of the maximum-density segment problem.

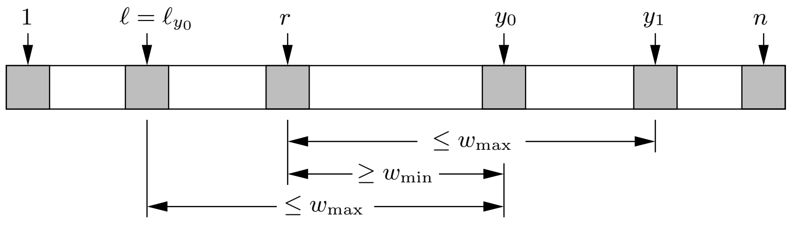


FIG. 4.1. *An illustration for the relation among* $\ell, r, y_0, y_1$.

### 4.1. A variant version of the maximum-density segment problem.

Suppose that we are given two indices $r$ and $y_0$ with $w(r, y_0) \geq w_{\min}$. Let $X = [\ell, r]$ and $Y = [y_0, y_1]$ be two intervals such that $\ell = \ell_{y_0}$ and $y_1$ is the largest index in $J$ with $w(r, y_1) \leq w_{\max}$. See Figure 4.1 for an illustration. The *variant version* of the maximum-density segment problem is to look for a maximum-density segment over

```
ALGORITHM VMAIN($r, y_0$).
    let $\ell$ be the smallest index in $[1, n]$ with $w(\ell, y_0) \leq w_{\max}$;
    let $y_1$ be the largest index in $[1, n]$ with $w(r, y_1) \leq w_{\max}$;
    let $x_{y_0-1} = \ell$;
    for $y = y_0$ to $y_1$ do {
        let $x_y$ = BEST($\max(x_{y-1}, \ell_y), r, y$);
        output $(x_y, y)$;
    }
```

FIG. 4.2. *Our algorithm for the variant version of the maximum-density segment problem, where function* BEST *is as defined in Figure* 2.2.

all feasible segments $S(x, y)$ with $x \in X$, $y \in Y$, and $w_{\min} \leq w(x, y) \leq w_{\max}$ such that $d(x, y)$ is maximized.

For each $y \in Y$, let $x^*_y$ be the largest index $x \in X$ with $w_{\min} \leq w(x, y) \leq w_{\max}$ that maximizes $d(x, y)$. Let $y^*$ be an index in $Y$ with $d(x^*_{y^*}, y^*) = \max_{y \in Y} d(x^*_y, y)$. Although solving the variant version can naturally be reduced to computing the index $x^*_y$ for each index $y \in Y$, the required running time is more than what we can afford. Instead, we compute an index $x_y \in X$ with $w_{\min} \leq w(x_y, y) \leq w_{\max}$ for each index $y \in Y$ such that $x_{y^*} = x^*_{y^*}$. By $w(r, y_0) \geq w_{\min}$ and $w(r, y_1) \leq w_{\max}$, one can easily see that, for each $y \in Y$, $r$ is always the largest index $x \in X$ with $w_{\min} \leq w(x, y) \leq w_{\max}$. Our algorithm for solving the variant problem is as shown in Figure 4.2, presented in a way to emphasize the analogy between VMAIN and MAIN. For example, the index $x_y$ in VMAIN is the counterpart of the index $i_j$ in MAIN. Also, the index $r$ in VMAIN plays the role of the index $r_j$ in MAIN. We have the following lemma whose proof is very similar to that of Theorem 2.2.

LEMMA 4.1. *Algorithm* VMAIN *solves the variant version of the maximum-density problem correctly.*

*Proof.* We prove the theorem by showing $x_{y^*} = x^*_{y^*}$. By $\ell_{y_0} = x_{y_0-1} = \ell$ and Lemma 2.1, the equality holds if $y^* = y_0$. The rest of the proof assumes $y^* > y_0$. By Lemma 2.1 and $\ell_{y^*} \leq x^*_{y^*}$, it suffices to ensure $x_{y^*-1} \leq x^*_{y^*}$. Assume for contradiction that there is an index $y \in [y_0, y^* - 1]$ with $x_{y-1} \leq x^*_{y^*} < x_y$. By $y < y^*$, we know $\ell_y \leq x^*_{y^*}$. By Lemma 2.1 and $\max(\ell_y, x_{y-1}) \leq x^*_{y^*} < x_y \leq r$, we have $d(x_y, y) \geq d(x^*_{y^*}, y)$. It follows from (1.1) and $x^*_{y^*} < x_y$ that $d(x^*_{y^*}, y) \geq d(x^*_{y^*}, x_y - 1)$. By $\ell_{y^*} \leq x^*_{y^*} < x_y \leq r$ and the definition of $y^*$, we know $d(x^*_{y^*}, y^*) > d(x_y, y^*)$. It follows from $x^*_{y^*} < x_y$ and (1.1) that $d(x^*_{y^*}, x_y - 1) > d(x^*_{y^*}, y^*)$. Therefore, $d(x_y, y) \geq d(x^*_{y^*}, y) \geq d(x^*_{y^*}, x_y - 1) > d(x^*_{y^*}, y^*)$, contradicting the definition of $y^*$. ∎

Again, the challenge lies in supporting each query to $\phi(i, r-1)$ of BEST in $O(1)$ time during the execution of VMAIN. Fortunately, unlike during the execution of MAIN, where both parameters of $\phi(i, r-1)$ may change, the second parameter $r - 1$ is now fixed. Therefore, to support each query to $\phi(i, r-1)$ in $O(1)$ time, we can actually afford $O(r - \ell + 1)$ time to compute a data structure $\Psi$ such that $\Psi[i] = \phi(i, r-1)$ for each $i \in [\ell, r-1]$. As a result, the function BEST can be implemented as the function VBEST shown in Figure 4.3. The following lemma ensures the correctness and efficiency of our implementation VARIANT shown in Figure 4.3.

LEMMA 4.2. *The implementation* VARIANT *correctly solves the variant version of the maximum-density segment problem in* $O(r - \ell + y_1 - y_0 + 1)$ *time.*

*Proof.* One can easily verify that if $\Psi[i] = \phi(i, r-1)$ holds for each index $i \in$

```
   ALGORITHM VARIANT($r, y_0$).
1      let $\ell$ be the smallest index in $[1, n]$ with $w(\ell, y_0) \le w_{\max}$;
2      let $y_1$ be the largest index in $[1, n]$ with $w(r, y_1) \le w_{\max}$;
3      call INIT($\ell, r - 1$);
4      let $x_{y_0 - 1} = \ell$;
5      for $y = y_0$ to $y_1$ do {
6          let $x_y$ = VBEST($\max(x_{y-1}, \ell_y), r, y$);
7          output $(x_y, y)$;
8      }

   FUNCTION VBEST($\ell, r, y$).
1      let $x = \ell$;
2      while $x < r$ and $d(x, \Psi[x]) \le d(x, y)$ do
3          let $x = \Psi[x] + 1$;
4      return $x$;

   SUBROUTINE INIT($\ell, r$)
1      let $\Psi[r] = r$;
2      for $s = r - 1$ downto $\ell$ do {
3          let $t = s$;
4          while $t < r$ and $d(s, t) \ge d(s, \Psi[t + 1])$ do
5              let $t = \Psi[t + 1]$;
6          let $\Psi[s] = t$;
7      }
```

FIG. 4.3. *An efficient implementation for the algorithm* VMAIN.

$[\ell, r-1]$, then VBEST is a faithful implementation of BEST. Therefore, by Lemma 4.1, the correctness of VARIANT can be ensured by showing that after calling INIT$(\ell, r-1)$ at step 3 of algorithm VARIANT , $\Psi[i] = \phi(i, r-1)$ holds for each index $i \in [\ell, r-1]$. Note that for brevity of the following proof, we slightly abuse the notation $r$ in Figure 4.3. That is, although the subroutine call at step 3 of algorithm VARIANT is INIT$(\ell, r-1)$, the second parameter in the definition of subroutine INIT in Figure 4.3 becomes $r$. Let us make it clear that the rest of the proof (i) lets $r$ denote the one in the definition of subroutine INIT$(\ell, r)$, and (ii) proves that $\Psi[i] = \phi(i, r)$ holds for each index $i \in [\ell, r]$ at the end of the subroutine call to subroutine INIT$(\ell, r)$.

By step 1 of INIT, we have $\Psi[r] = r = \phi(r, r)$. Now suppose that $\Psi[i] = \phi(i, r)$ holds for each index $i \in [x+1, r]$ right before INIT is about to execute the iteration for index $x \in [\ell, r]$. It suffices to show $\Psi[x] = \phi(x, r)$ after the iteration. Let

$$Z_x = \{x, \Psi[x+1], \Psi[\Psi[x+1]+1], \Psi[\Psi[\Psi[x+1]+1]+1], \ldots, r\}.$$

Let $|Z_x|$ denote the cardinality of $Z_x$. We first show $\phi(x, r) \in Z_x$ as follows.

Assume for contradiction that $\phi(x, r) \notin Z_x$, i.e., there is an index $z \in Z_x$ with $z < \phi(x, r) < \Psi[z+1] = \phi(z+1, r)$. By definition of $\phi$ and (1.1), we have $d(z+1, \phi(x, r)) > d(z+1, \phi(z+1, r)) > d(\phi(x, r)+1, \phi(z+1, r))$ and $d(\phi(x, r)+1, \phi(z+1, r)) \ge d(x, \phi(z+1, r)) \ge d(x, \phi(x, r))$. By $d(z+1, \phi(x, r)) > d(x, \phi(x, r))$ and (1.1), we have $d(x, \phi(x, r)) > d(x, z)$, contradicting the definition of $\phi(x, r)$.

For any index $z \in Z_x$ with $z < \phi(x, r)$, we know $z < r$ and $\phi(z+1, r) = \Psi[z+1] \le$

```
ALGORITHM GENERAL.
1      let $p = q = r_{j_0-1} = \Phi[1] = 1$;
2      for $j = j_0$ to $n$ do {
3          call UPDATE($j$);
4          while $\Phi[p] < \ell_j$ do
5              let $p = p + 1$;
6          if $i_{j-1} < \Phi[p]$ then
7              call VARIANT($\Phi[p], j$);
8          let $i_j$ = LBEST($j$);
9          output $(i_j, j)$;
10     }
```

FIG. 4.4. *Our algorithm for the general case, where* UPDATE *and* LBEST *are defined in Figure* 3.2 *and* VARIANT *is defined in Figure* 4.3.

$\phi(x, r)$. By $\phi(z + 1, r) \le \phi(x, r) \le r$ and the definition of $\phi(z + 1, r)$, we have $d(z + 1, \phi(x, r)) \ge d(z + 1, \phi(z + 1, r))$. By definition of $\phi(x, r)$ and (1.1), we have $d(x, z) \ge d(x, \phi(x, r)) \ge d(z + 1, \phi(x, r))$. By $d(x, z) \ge d(z + 1, \phi(z + 1, r))$ and (1.1), we have $d(x, z) \ge d(x, \phi(z + 1, r))$. Therefore, if $z < \phi(x, r)$, then step 5 of INIT will be executed to increase the value of $z$. Observe that $\phi(x, r) = z < r$ and $\Psi[z + 1] > z$ imply $d(x, z) < d(x, \Phi[z + 1])$. It follows that as soon as $z = \phi(x, r)$ holds, whether $\phi(x, r) = r$ or not, the value of $\Psi[x]$ will immediately be set to $z$ at step 6 of INIT.

As for the time complexity, we first observe that $\ell$ and $y_1$ can be found from $r$ and $y_0$ in $O(r - \ell + y_1 - y_0 + 1)$ time:

- Let $\ell = r$, and then repeatedly decrease $\ell$ by 1 as long as $w(\ell - 1, y_0) \le w_{\max}$ holds.
- Let $y_1 = y_0$, and then repeatedly increase $y_1$ by 1 as long as $w(r, y_1 + 1) \le w_{\max}$ holds.

Secondly, one can see that the rest of the implementation also runs in $O(r - \ell + y_1 - y_0 + 1)$ time by verifying that throughout the execution of the implementation (a) the while-loop of VBEST runs for $O(r - \ell + y_1 - y_0 + 1)$ iterations, and (b) the while-loop of INIT runs for $O(r - \ell + 1)$ iterations. To see statement (a), just observe that the value of index $x$ (i) never decreases, (ii) stays in $[\ell, r]$, and (iii) increases by at least one each time step 3 of VBEST is executed. As for statement (b), consider the iteration with index $s$ of the for-loop of INIT. Note that if step 6 of INIT executes $t_s$ times in this iteration, then $|Z_s| = |Z_{s+1}| - t_s + 1$. Since $|Z_s| \ge 1$ holds for each $s \in X$, we have $\sum_{s \in X} t_s = O(r - \ell + 1)$, and thus statement (b) holds. □

**4.2. Our algorithm for the general case.** With the help of VARIANT, we have a linear-time algorithm for solving the original maximum-density segment problem as shown in Figure 4.4. Algorithm GENERAL is obtained by inserting four lines of code (i.e., steps 4–7 of GENERAL) between steps 3 and 4 of LMAIN in order to handle the case $i_{j-1} < \ell_j$. Specifically, when $i_{j-1} < \ell_j$, we cannot afford to appropriately update the data structure $\Phi$. Therefore, instead of moving $i$ to $\ell_j$, steps 4 and 5 move $i$ to $\Phi[p]$, where $p$ is the smallest index with $\ell_j \le \Phi[p]$. Of course, these two steps may cause our algorithm to overlook the possibility of $i_j \in [i_{j-1}, \Phi[p] - 1]$, as illustrated in Figure 4.5. This is when the variant version comes in: As shown in the next theorem, we can remedy the problem by calling VARIANT($\Phi[p], j$).

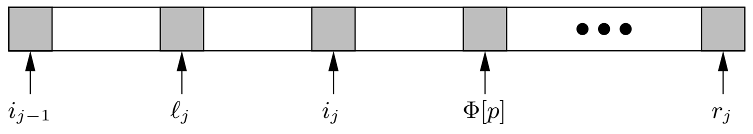


FIG. 4.5. *An illustration for the situation when Steps* 6 *and* 7 *of* GENERAL *are needed.*

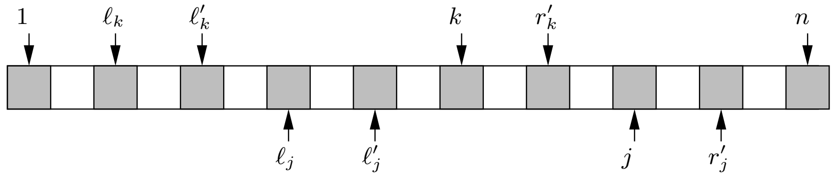


FIG. 4.6. *An illustration showing that the overall running time of all subroutine calls to* VARIANT$(\ell'_j, j)$ *in* GENERAL *is* $O(n)$.

THEOREM 4.3. *Algorithm* GENERAL *solves the maximum-density segment problem in an online manner in linear time.*

*Proof.* We prove the correctness of GENERAL by showing that $i^*_{j^*} \neq i_{j^*}$ implies $i^*_{j^*} = x_{j^*}$. By Lemma 3.1(1), Condition $C_j$ holds after the subroutine call UPDATE$(j)$ at step 3 of GENERAL. Observe that steps 4 and 5 of GENERAL, which may increase the value of $p$, do not affect the validity of Condition $C_j$. Also, steps 6 and 7 do not modify $p$, $q$, and $\Phi$. Let $\ell'_j$ be the value of $\Phi[p]$ right before executing step 8 of GENERAL. By Lemma 3.1(2), the index $i_j$ returned by LBEST$(j)$ is the largest index in $[\ell'_j, r_j]$ that maximizes $d(i_j, j)$. Clearly, $i_{j^*} = i^*_{j^*}$ implies the correctness of GENERAL. If $i_{j^*} \neq i^*_{j^*}$, there must be an index $j \in [j_0, j^*]$ with $i_{j-1} \leq i^*_{j^*} < i_j$. It can be proved as follows that $i^*_{j^*} \leq \ell'_j - 1$.

> Assume $\ell'_j \leq i^*_{j^*}$ for contradiction. It follows from Lemma 3.1(2) and (1.1) that $d(i_j, j) \geq d(i^*_{j^*}, j) \geq d(i^*_{j^*}, i_j - 1)$. By the definition of $j^*$, we have $d(i^*_{j^*}, j^*) > d(i_j, j^*)$, which by (1.1) implies $d(i^*_{j^*}, i_j - 1) > d(i^*_{j^*}, j^*)$. Therefore, $d(i_j, j) > d(i^*_{j^*}, j^*)$, contradicting the definition of $j^*$.

Since $i_{j-1} \leq i^*_{j^*} \leq \ell'_j - 1$, we know $w(\ell'_j - 1, j^*) \leq w(i^*_{j^*}, j^*) \leq w_{\max}$. Thus, $S(i^*_{j^*}, j^*)$ is a feasible segment in the variant version of the maximum-density segment problem for $S$ with respect to indices $r = \ell'_j$ and $y_0 = j$. By Lemma 4.2, the subroutine call VARIANT$(\ell'_j, j)$ at step 7 of algorithm GENERAL has to output an index pair $(x, y)$ with $w_{\min} \leq w(x, y) \leq w_{\max}$ and $d(x, y) = d(i^*_{j^*}, j^*)$.

As for the running time, observe that $q - p \geq 0$ holds throughout the execution of GENERAL. Step 4 of UPDATE, which is the only place that increases the value of $q - p$, increases the value of $q - p$ by 1 for $O(n)$ times. Note that each iteration of the while-loops of GENERAL, LBEST, and UPDATE decreases the value of $q - p$ by 1. Therefore, to show that the overall running time of GENERAL is $O(n)$, it remains to ensure that all those subroutine calls to VARIANT at step 7 of GENERAL take overall $O(n)$ time. Suppose that $j$ and $k$ are two arbitrary indices with $k < j$ such that GENERAL makes subroutine calls to VARIANT$(\ell'_k, k)$ and VARIANT$(\ell'_j, j)$. Let $r'_k$ be the largest index in $[1, n]$ with $w(\ell'_k, r'_k) \leq w_{\max}$. By Lemma 4.2, it suffices to show $\ell'_k < \ell_j$ and $r'_k < j$ as follows. (See Figure 4.6.) By the definition of GENERAL, we know that $i_{j-1} < \ell_j$, which is ensured by the situation illustrated in Figure 4.5. By $k < j$, we have $\ell'_k \leq i_{j-1}$, implying $\ell'_k < \ell_j$. Moreover, by the definitions of $\ell_j$ and

$r'_k$, one can easily verify that $\ell'_k < \ell_j$ implies $r'_k < j$.

It is not difficult to see that our algorithm shown in Figure 4.4 is already capable of processing the input sequence in an online manner, since the only preprocessing required is to obtain $\ell_j$, $r_j$, and the prefix sums of $a_1, a_2, \dots, a_j$ and $w_1, w_2, \dots, w_j$ (for the purpose of evaluating the density of any segment in $O(1)$ time), which can easily be computed on the fly. ∎

## 5. Exploiting sparsity for the uniform case.

In this section, we assume that the input sequence $S = (a_1, w_1), (a_2, w_2), \dots, (a_n, w_n)$ is run-length encoded [15], i.e., represented by $m$ pairs $(a'_1, n_1), (a'_2, n_2), \dots, (a'_m, n_m)$ with $0 = n_0 < n_1 < n_2 < \cdots < n_m = n$ to signify that $w_1 = w_2 = \cdots = w_n = 1$ and that $a_i = a'_j$ holds for all indices $i$ and $j$ with $n_{j-1} < i \le n_j$ and $1 \le j \le m$. Our algorithm for solving the maximum-density problem for the $O(m)$-space representable sequence $S$ is shown in Figure 5.1.

```
Algorithm sparse.
1    for $k = 1$ to $m$ do
2        let $n'_k = n_k - n_{k-1}$;
3    let $S'$ be the length-$m$ sequence $(n'_1 a'_1, n'_1), (n'_2 a'_2, n'_2), \dots, (n'_m a'_m, n'_m)$;
4    let $(i', j')$ be an optimal output of general$(w_{\min}, w_{\max}, S')$;
5    output $(n_{i'-1} + 1, n_{j'})$;
6    for $k = 1$ to $m$ do {
7        if $n_k \ge w_{\min}$ then
8            output $(\ell_{n_k}, n_k)$ and $(r_{n_k}, n_k)$;
9        if $n_{k-1} + w_{\min} \le n$ then
10           output $(n_{k-1} + 1, n_{k-1} + w_{\min})$ and $(n_{k-1} + 1, \min(n, n_{k-1} + w_{\max}))$;
11   }
```

Fig. 5.1. *Our algorithm that handles sparse input sequence for the uniform case, where* general *is defined in Figure* 4.4.

Theorem 5.1. *Algorithm* sparse *solves the maximum-density problem for the above $O(m)$-space representable sequence in $O(m)$ time.*

*Proof.* By Theorem 4.3, sparse runs in $O(m)$ time. Let $S(i^*, j^*)$ be a feasible segment with maximum density. We first show that without loss of generality $i^* - 1 \in \{n_0, n_1, \dots, n_{m-1}\}$ or $j^* \in \{n_1, n_2, \dots, n_m\}$ holds. More specifically, we show as follows that if $i^* - 1 \notin \{n_0, n_1, \dots, n_{m-1}\}$ and $j^* \notin \{n_1, n_2, \dots, n_m\}$, then $S(i^* + 1, j^* + 1)$ is also a feasible segment with maximum density.

> By $i^* - 1 \notin \{n_0, n_1, \dots, n_{m-1}\}$, we know $a_{i^*-1} = a_{i^*}$. By $j^* \notin \{n_1, n_2, \dots, n_m\}$, we know $a_{j^*} = a_{j^*+1}$. It follows from the optimality of $S(i^*, j^*)$ that $a_{i^*} \ge a_{j^*+1}$ and $a_{i^*-1} \le a_{j^*}$, implying $a_{i^*-1} = a_{i^*} = a_{j^*} = a_{j^*+1}$. Therefore, $S(i^* + 1, j^* + 1)$ is also a maximum-density segment.

It remains to show that our algorithm works correctly for each possible case.

- Case 1: $i^* - 1 \in \{n_0, n_1, \dots, n_{m-1}\}$ and $j^* \in \{n_1, n_2, \dots, n_m\}$. Clearly, steps 1–5 of sparse take care of this case.
- Case 2: $i^* - 1 \notin \{n_0, n_1, \dots, n_{m-1}\}$ and $j^* \in \{n_1, n_2, \dots, n_m\}$. Clearly, if $i^* \in \{\ell_{j^*}, r_{j^*}\}$, $S(i^*, j^*)$ can be discovered by steps 7 and 8 of sparse. Since $i^* - 1 \notin \{n_0, n_1, \dots, n_{m-1}\}$, we have $a_{i^*-1} = a_{i^*}$. If $a_{i^*-1} = a_{i^*} \ne d(i^*, j^*)$, then by (1.1) we have either $d(i^*-1, j^*) > d(i^*, j^*)$ or $d(i^*+1, j^*) > d(i^*, j^*)$,

which implies that either $S(i^*-1,j^*)$ or $S(i^*+1,j^*)$ is infeasible, and thus $i^* \in \{\ell_{j^*}, r_{j^*}\}$. On the other hand, if $a_{i^*-1} = a_{i^*} = d(i^*,j^*)$ and $i^* \neq \ell_j^*$, then $S(i^*-1,j^*)$ is also a feasible segment with maximum density. We can continue the same argument until we have a maximum-density segment $S(i,j^*)$ such that either $i-1 \in \{n_0, n_1, \ldots, n_{m-1}\}$, which is handled in Case 1, or $i = \ell_{j^*}$, which is handled by steps 7 and 8 of SPARSE.

- Case 3: $i^*-1 \in \{n_0, n_1, \ldots, n_{m-1}\}$ and $j^* \notin \{n_1, n_2, \ldots, n_m\}$. The proof of this case, omitted for brevity, is very similar to that of Case 2.

The theorem is proved. □

**Acknowledgments.** We thank Yi-Hsuan Hsin and Hsu-Cheng Tsai for discussions during the preliminary stage of this research. We thank Tien-Ching Lin for comments.

## REFERENCES

[1] N. N. Alexandrov and V. V. Solovyev, *Statistical significance of ungapped sequence alignments*, in Proceedings of the Pacific Symposium on Biocomputing, Maui, HI, 1998, World Scientific, River Edge, NJ, Vol. 3, pp. 461–470.

[2] G. Barhardi, *Isochores and the evolutionary genomics of vertebrates*, Gene, 241 (2000), pp. 3–17.

[3] G. Bernardi and G. Bernardi, *Compositional constraints and genome evolution*, J. Molec. Evolution, 24 (1986), pp. 1–11.

[4] H. Brönnimann and B. Chazelle, *Optimal slope selection via cuttings*, Comput. Geom., 10 (1998), pp. 23–29.

[5] B. Charlesworth, *Genetic recombination: Patterns in the genome*, Current Biol., 4 (1994), pp. 182–184.

[6] R. Cole, J. S. Salowe, W. L. Steiger, and E. Szemerédi, *An optimal-time algorithm for slope selection*, SIAM J. Comput., 18 (1989), pp. 792–810.

[7] L. Duret, D. Mouchiroud, and C. Gautier, *Statistical analysis of vertebrate sequences reveals that long genes are scarce in GC-rich isochores*, J. Mol. Evol., 40 (1995), pp. 308–371.

[8] A. Eyre-Walker, *Evidence that both G+C rich and G+C poor isochores are replicated early and late in the cell cycle*, Nucleic Acids Res., 20 (1992), pp. 1497–1501.

[9] A. Eyre-Walker, *Recombination and mammalian genome evolution*, Proc. Roy. Soc. London Ser. B, 252 (1993), pp. 237–243.

[10] J. Filipski, *Correlation between molecular clock ticking, codon usage fidelity of DNA repair, chromosome banding and chromatin compactness in germline cells*, FEBS Lett., 217 (1987), pp. 184–186.

[11] M. P. Francino and H. Ochman, *Isochores result from mutation not selection*, Nature, 400 (1999), pp. 30–31.

[12] S. M. Fullerton, A. B. Carvalho, and A. G. Clark, *Local rates of recombination are positively correlated with GC content in the human genome*, Mol. Biol. Evol., 18 (2001), pp. 1139–1142.

[13] M. H. Goldwasser, M.-Y. Kao, and H.-I. Lu, *Fast algorithms for finding maximum-density segments of a sequence with applications to bioinformatics*, in Proceedings of the Second International Workshop on Algorithms in Bioinformatics (Rome, Italy, 2002), R. Guigó and D. Gusfield, eds., Lecture Notes in Comput. Sci. 2452, Springer-Verlag, New York, 2002, pp. 157–171.

[14] M. H. Goldwasser, M.-Y. Kao, and H.-I. Lu, *Linear-time algorithms for computing maximum-density sequence segments with bioinformatics applications*, J. Comput. System Sci., to appear.

[15] S. W. Golomb, *Run-length encodings*, IEEE Trans. Inform. Theory, 12 (1966), pp. 399–401.

[16] P. Guldberg, K. Gronbak, A. Aggerholm, A. Platz, P. thor Straten, V. Ahrenkiel, P. Hokland, and J. Zeuthen, *Detection of mutations in GC-rich DNA by bisulphite denaturing gradient gel electrophoresis*, Nucleic Acids Res., 26 (1998), pp. 1548–1549.

[17] W. Henke, K. Herdel, K. Jung, D. Schnorr, and S. A. Loening, *Betaine improves the PCR amplification of GC-rich DNA sequences*, Nucleic Acids Res., 25 (1997), pp. 3957–3958.

[18] G. P. Holmquist, *Chromosome bands, their chromatin flavors, and their functional features*, Am. J. Hum. Genet., 51 (1992), pp. 17–37.

[19] X. Huang, *An algorithm for identifying regions of a DNA sequence that satisfy a content requirement*, Comput. Appl. Biosci., 10 (1994), pp. 219–225.

[20] K. Ikehara, F. Amada, S. Yoshida, Y. Mikata, and A. Tanaka, *A possible origin of newly born bacterial genes: Significance of GC-rich nonstop frame on antisense strand*, Nucleic Acids Res., 24 (1996), pp. 4249–4255.

[21] R. B. Inman, *A denaturation map of the* 1 *phage DNA molecule determined by electron microscopy*, J. Mol. Biol., 18 (1966), pp. 464–476.

[22] I. P. Ioshikhes and M. Q. Zhang, *Large-scale human promoter mapping using CpG islands*, Nature Genet., 26 (2000), pp. 61–63.

[23] R. Jin, M.-E. Fernandez-Beros, and R. P. Novick, *Why is the initiation nick site of an AT-rich rolling circle plasmid at the tip of a GC-rich cruciform?*, EMBO J., 16 (1997), pp. 4456–4466.

[24] M. J. Katz and M. Sharir, *Optimal slope selection via expanders*, Inform. Process. Lett., 47 (1993), pp. 115–122.

[25] S. K. Kim, *Linear-time algorithm for finding a maximum-density segment of a sequence*, Inform. Process. Lett., 86 (2003), pp. 339–342.

[26] Y.-L. Lin, X. Huang, T. Jiang, and K.-M. Chao, *MAVG: Locating nonoverlapping maximum average segments in a given sequence*, Bioinformatics, 19 (2003), pp. 151–152.

[27] Y.-L. Lin, T. Jiang, and K.-M. Chao, *Algorithms for locating the length-constrained heaviest segments, with applications to biomolecular sequence analysis*, J. Comput. System Sci., 65 (2002), pp. 570–586.

[28] G. Macaya, J.-P. Thiery, and G. Bernardi, *An approach to the organization of eukaryotic genomes at a macromolecular level*, J. Mol. Biol., 108 (1976), pp. 237–254.

[29] C. S. Madsen, C. P. Regan, and G. K. Owens, *Interaction of CArG elements and a GC-rich repressor element in transcriptional regulation of the smooth muscle myosin heavy chain gene in vascular smooth muscle cells*, J. Biol. Chem., 272 (1997), pp. 29842–29851.

[30] J. Matoušek, *Randomized optimal algorithm for slope selection*, Inform. Process. Lett., 39 (1991), pp. 183–187.

[31] S.-I. Murata, P. Herman, and J. R. Lakowicz, *Texture analysis of fluorescence lifetime images of AT- and GC-rich regions in nuclei*, J. Histochem. Cytochem., 49 (2001), pp. 1443–1452.

[32] A. Nekrutenko and W.-H. Li, *Assessment of compositional heterogeneity within and between eukaryotic genomes*, Genome Res., 10 (2000), pp. 1986–1995.

[33] U. Ohler, H. Niemann, G. Liao, and G. M. Rubin, *Joint modeling of DNA sequence and physical properties to improve eukaryotic promoter recognition*, Bioinformatics, 17 (2001), pp. S199–S206.

[34] P. Rice, I. Longden, and A. Bleasby, *EMBOSS: The European molecular biology open software suite*, Trends Genet., 16 (2000), pp. 276–277.

[35] L. Scotto and R. K. Assoian, *A GC-rich domain with bifunctional effects on mRNA and protein levels: Implications for control of transforming growth factor beta 1 expression*, Mol. Cell. Biol., 13 (1993), pp. 3588–3597.

[36] P. H. Sellers, *Pattern recognition in genetic sequences by mismatch density*, Bull. Math. Biol., 46 (1984), pp. 501–514.

[37] P. M. Sharp, M. Averof, A. T. Lloyd, G. Matassi, and J. F. Peden, *DNA sequence evolution: The sounds of silence*, Philos. Trans. Soc. Lond. B Biol. Sci., 349 (1995), pp. 241–247.

[38] P. Soriano, M. Meunier-Rotival, and G. Bernardi, *The distribution of interspersed repeats is nonuniform and conserved in the mouse and human genomes*, Proc. Natl. Acad. Sci. USA, 80 (1983), pp. 1816–1820.

[39] N. Stojanovic, L. Florea, C. Riemer, D. Gumucio, J. Slightom, M. Goodman, W. Miller, and R. Hardison, *Comparison of five methods for finding conserved sequences in multiple alignments of gene regulatory regions*, Nucleic Acids Res., 27 (1999), pp. 3899–3910.

[40] N. Sueoka, *Directional mutation pressure and neutral molecular evolution*, Proc. Natl. Acad. Sci. USA, 80 (1988), pp. 1816–1820.

[41] Z. Wang, E. Lazarov, M. O'Donnel, and M. F. Goodman, *Resolving a fidelity paradox: Why Escherichia coli DNA polymerase* II *makes more base substitution errors in At- compared to GC-rich DNA*, J. Biol. Chem., 277 (2002), pp. 4446–4454.

[42] K. H. Wolfe, P. M. Sharp, and W.-H. Li, *Mutation rates differ among regions of the mammalian genome*, Nature, 337 (1989), pp. 283–285.

[43] Y. Wu, R. P. Stulp, P. Elfferich, J. Osinga, C. H. Buys, and R. M. Hofstra, *Improved mutation detection in GC-rich DNA fragments by combined DGGE and CDGE*, Nucleic

Acids Res., 27 (1999), article e9.
[44] S. Zoubak, O. Clay, and G. Bernardi, *The gene distribution of the human genome*, Gene, 174 (1996), pp. 95–102.

SIAM J. COMPUT. © 2004 Society for Industrial and Applied Mathematics
Vol. 34, No. 2, pp. 388–404

# NEW APPROXIMATION TECHNIQUES FOR SOME LINEAR ORDERING PROBLEMS*

SATISH RAO† AND ANDRÉA W. RICHA‡

**Abstract.** We describe logarithmic approximation algorithms for the NP-hard graph optimization problems of minimum linear arrangement, minimum containing interval graph, and minimum storage–time product. This improves upon the best previous approximation bounds of Even, Naor, Rao, and Schieber [*J. ACM*, 47 (2000), pp. 585–616] for these problems by a factor of $\Omega(\log\log n)$.

We use the lower bound provided by the volume $W$ of a spreading metric for each of the ordering problems above (as defined by Even et al.) in order to find a solution with cost at most a *logarithmic factor times* $W$ for these problems. We develop a divide-and-conquer strategy where the cost of a solution to a problem at a recursive level is $C$ plus the cost of a solution to the subproblems at this level, *and* where the spreading metric volume on the subproblems is less than the original volume by $\Omega(C/\log n)$, ensuring that the resulting solution has cost $O(\log n)$ times the original spreading metric volume. We note that this is an existentially tight bound on the relationship between the spreading metric volume and the true optimal values for these problems.

For planar graphs, we combine a structural theorem of Klein, Plotkin, and Rao [*Proceedings of the* 25th *ACM Symposium on Theory of Computing*, 1993, pp. 682–690] with our new recursion technique to show that the spreading metric cost volumes are within an $O(\log\log n)$ factor of the cost of an optimal solution for the minimum linear arrangement, and the minimum containing interval graph problems.

**Key words.** minimum linear arrangement, interval graph completion, storage–time product, spreading metrics, approximation algorithms

**AMS subject classifications.** 68W40, 68W25, 68Q25

**DOI.** 10.1137/S0097539702413197

## 1. Introduction.

We describe the approximation algorithms that apply to the NP-hard graph optimization problems of finding a minimum linear arrangement, a minimum containing interval graph, and a minimum storage–time product [5]. All these problems can be viewed as linear ordering problems. In a linear ordering problem on a graph $G$, the nodes of $G$ need to be ordered linearly—i.e., from $1, \ldots, n$—so as to minimize (or maximize) some given function of the ordering. An $\alpha$-*approximation algorithm* is an algorithm that finds a solution to the respective problem whose cost is at most $\alpha$ times the cost of an optimal solution to the problem.

All the ideas that we use for approximating the minimum containing interval graph and the minimum storage–time product problems can be illustrated by the algorithms for the minimum linear arrangement problem. Thus, we restrict our exposition primarily to the minimum linear arrangement problem, which we define as follows: Let $G$ be a graph with associated edge weights. Informally, a minimum linear arrangement (MLA) of $G$ is an embedding[1] of $G$ in the linear array such that (i) we

*Received by the editors August 19, 2002; accepted for publication (in revised form) December 3, 2003; published electronically December 1, 2004. A preliminary version of this work appeared in *Proceedings of the Ninth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 1998.
http://www.siam.org/journals/sicomp/34-2/41319.html

†Computer Science Division, University of California, Soda Hall, Berkeley, CA 94720-1776 (satishr@cs.berkeley.edu).

‡Department of Computer Science and Engineering, Arizona State University, Tempe, AZ 85287 (aricha@asu.edu). The work of this author was supported in part by NSF CAREER award CCR-9985284 and NSF grant CCR-9900304.

[1]An *embedding* of a graph $G$ into a graph $H$ maps nodes of $G$ to nodes of $H$, and edges of $G$ to paths in $H$. Typically, a guest network $G$ is emulated by a host network $H$ by embedding $G$ into $H$. (For a more complete discussion of emulations and embeddings, see [9].)

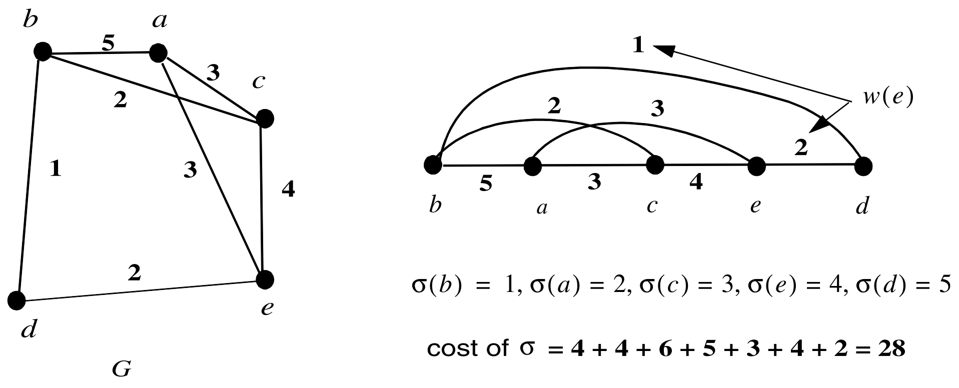

b
a
c
d
e
G
5
3
2
1
3
4
2
1
2
3
2
w(e)
b
a
c
e
d
5
3
4
σ(b) = 1, σ(a) = 2, σ(c) = 3, σ(e) = 4, σ(d) = 5
cost of σ = 4 + 4 + 6 + 5 + 3 + 4 + 2 = 28

FIG. 1. *A graph $G$ and an MLA $\sigma$ of $G$.*

have a one-to-one mapping from the nodes of $G$ to the nodes of the linear array, and (ii) the weighted sum of the lengths of the edges of $G$—that is, the *cost* of the linear arrangement—is minimum. The length of an edge of $G$ in the embedding is given by the distance between its two endpoints on the linear array. In Figure 1, we show a linear arrangement $\sigma$ for the graph $G$ with cost 28 (in fact this linear arrangement is an MLA of $G$).

Finding an MLA is NP-hard, even for the case when all the edges have unit weight. We present a polynomial time $O(\log n)$-approximation algorithm for the MLA problem on a graph with $n$ nodes. This improves the best previous approximation bound of Even, Naor, Rao, and Schieber [3] for this problem by a factor of $O(\log \log n)$.

We extend our approximation techniques (and bounds) to two other problems that involve finding a linear ordering of the nodes of a graph: the minimum containing interval graph and the minimum storage–time product problems. Using techniques from [14], we can view the minimum containing interval graph problem as a "node version" of the MLA (see [3]). Thus, we also obtain an $O(\log n)$-approximation algorithm for the minimum containing interval graph problem on general graphs. This improves on the previous best-known bound of $O(\log n \log \log n)$ [3].

We can also use techniques from [14] to extend our ideas to produce an $O(\log T)$-approximation for the minimum storage–time product problem (where $T$ is the sum of the processing times of all tasks), improving on a previous approximation bound of $O(\log T \log \log T)$ in [3]. The minimum storage–time product problem can be viewed as a generalization of the MLA, as explained in section 4.

If the graph is planar (or, more generally, if it excludes $K_{r,r}$ as a minor, for fixed $r$, where $K_{r,r}$ is the $r \times r$ complete bipartite graph), we obtain an $O(\log \log n)$-approximation factor for the MLA problem—improving upon the best-known bound of $O(\log n)$ for these graphs—using a variation of the algorithm presented for the general case. We obtain this improvement by combining the techniques used for the general case with the algorithm presented by Klein, Plotkin, and Rao [8] for finding separators in graphs that exclude fixed $K_{r,r}$-minors.

Since we view the minimum containing interval graph problem as a "node variation" of the MLA problem, we are also able to obtain the same improved approximation bound of $O(\log \log n)$ for the minimum containing interval graph problem for graphs that exclude fixed $K_{r,r}$-minors, improving on the previous bound of $O(\log n)$. Note that the decomposition techniques of Klein, Plotkin, and Rao do not apply for directed graphs, and therefore do not yield a better approximation factor for the minimum storage–time product when restricted to the class of $K_{r,r}$-excluded minor graphs.

A small variation of our algorithm (as presented in [1]) can match the best-known existing approximation algorithms for two other related problems: Namely, a small variation of the algorithms presented in section 3 provides $O(\log^2 n)$-approximation for the minimum-cut linear arrangement and the minimum pathwidth problems.

Our approximation techniques rely on a lower bound $W$ on the cost of an optimal solution provided by a *spreading metric* (to be defined soon) for each of the problems considered: We find a solution to the general problem that has cost $O(W \log n)$ ($O(W \log T)$ for the minimum storage–time product problem). In [15], Seymour credits Alon with proving that there exists a logarithmic gap between the spreading metric cost volume and the true optimal cost for certain instances of the problem of finding a minimum feedback arc set. Alon's proof can be translated to prove analogous logarithmic gap bounds for the problems of MLA, minimum containing interval graph, and minimum storage–time product. Thus we provide an existentially tight bound on the relationship between the spreading metric cost volumes and the true optimal values for these problems. We briefly describe the approach for obtaining this lower bound in section 2.

**1.1. Previous work.** Leighton and Rao [10] presented an $O(\log n)$-approximation algorithm for balanced partitions of graphs. Among other applications, this provided $O(\log^2 n)$-approximation algorithms for the minimum feedback arc set and for the minimum-cut linear arrangement problem. Hansen [6] used the ideas in [10] to present $O(\log^2 n)$-approximation algorithms for the minimum linear arrangement problem and for the more general problem of graph embeddings in $d$-dimensional meshes. Ravi, Agrawal, and Klein [14] presented polynomial time approximation algorithms that deliver a solution with cost within an $O(\log n \log T)$ factor from optimal for the minimum storage–time product problem, where $T$ is the sum of the processing times of all tasks, and within an $O(\log^2 n)$ factor from optimal for the minimum containing interval graph.

Seymour [15] was the first to present a directed graph decomposition divide-and-conquer approach that does not rely on balanced cuts. He presented a polynomial time $O(\log n \log\log n)$-approximation algorithm for the minimum feedback arc set problem. Even et al. [3] extended the recursive decomposition technique used by Seymour to obtain polynomial time $O(\log n \log\log n)$-approximation algorithms for the MLA and the minimum containing interval graph problems, and an $O(\log T \log\log T)$-approximation algorithm for the minimum storage–time product problem. Even et al. actually showed similar approximation results for a broader class of graph optimization problems, namely, the ones that satisfy their "approximation paradigm": A graph optimization problem where their divide-and-conquer approach is applicable, and for which a spreading metric exists, satisfies this paradigm. They presented polynomial time $O(\min\{\log W \log\log W, \log k \log\log k\})$-approximation algorithms for these problems, where $k$ denotes the number of "interesting" nodes in the problem instance (clearly $k \leq n$), and $W$ is a lower bound on the cost of an optimal solution for the optimization problem provided by a spreading metric. Examples of such problems, besides the ones already mentioned, are graph embeddings in $d$-dimensional meshes, symmetric multicuts in directed networks, and multiway separators and $\rho$-separators (for small values of $\rho$) in directed graphs. For a detailed description of each of those problems, see [3].

Even et al. [2] extended the spreading metric techniques to graph partitioning problems. They used simpler recursions that yield a logarithmic approximation factor for balanced cuts and multiway separators. However, they were not able to extend this simpler technique to obtain a logarithmic approximation bound for the other problems considered in [3].

Recently, Bornstein and Vempala [1] proposed an alternative unified approach for obtaining lower bounds for the problems considered in this work. They present a unified linear program framework which defines "flow metrics" for each of the problems considered. Flow metrics are approximately equivalent to spreading metrics, in the sense that the spreading polyhedron is a projection of the flow distance polyhedron. While spreading metrics have exponentially many constraints, the definition of flow metrics uses more variables but only polynomially many constraints. By simply varying the objective function of the linear program, they obtain lower bounds for each of the problems, which they use in conjunction with the divide-and-conquer algorithm presented in this work (and its preliminary version in [13]) to match our approximation factors for the three problems considered. They also present a minor variation of our algorithm which can be used to obtain $O(\log^2 n)$-approximations for the minimum-cut linear arrangement and minimum pathwidth problems.

**1.2. Spreading metrics and our recursion.** Our algorithms use an approach that relies on *spreading metrics*. Spreading metrics have been used in recent divide-and-conquer techniques to obtain improved approximation algorithms for several graph optimization problems that are NP-hard [3]. These techniques perform the divide step according to the *cost* of a solution to the subproblems generated, rather than according to the *size* of such subproblems.

A spreading metric on a graph is an assignment of lengths to the edges or nodes of the graph that has the property of "spreading apart" (with respect to the metric lengths) all the nontrivial connected subgraphs. The *volume* of the spreading metric is the sum, taken over all edges (resp., nodes), of the length of each edge (resp., node) multiplied by its weight. For each of the optimization problems considered in this paper, Even et al. [3] showed how to find a spreading metric of volume $W$ such that $W$ is a lower bound on the cost of a solution to the problem. Our techniques are based on showing that a spreading metric of volume $W$ can be used to find a solution to the respective problem with cost $O(W \log n)$ ($O(W \log T)$ for the minimum storage–time product problem).

All of the spreading metrics used in this paper can be viewed as *one-dimensional spreadings metrics.* The main idea of a one-dimensional spreading metric is that the sum of the pairwise distances of any subset of $k$ nodes in the graph is at least the sum of the pairwise distances in a linear metric for $k$ uniformly spaced points on the line.

In this paper, we develop a recursion where at each level we identify cost which, if incurred, yields subproblems with reduced spreading metric volume. Specifically, we present a divide-and-conquer strategy where the cost of a solution to a problem at a recursive level is $C$ plus the cost of a solution to the subproblems, *and* where the spreading metric volume on the subproblems generated is less than the original volume by $\Omega(C/\log n)$ (resp., $\Omega(C/\log T)$ for the minimum storage–time product problem). We will show that this ensures that the resulting solution has cost $O(\log n)$ (resp., $O(\log T)$) times the original spreading metric volume.

The recursion is based on divide-and-conquer—that is, we find an edge or node set whose removal divides the graphs into subgraphs, and then recursively order the subgraphs. The cost of a recursive level is the cost associated with the edges (or nodes) in the cut selected at this level. Previous recursive methods and analyses proceeded by finding a small cutset where the maximum spreading metric volumes of the subproblems were quickly reduced. We proceed by finding a *sequence* of cutsets whose total cost can be upper bounded, say by a quantity $C$, and whose total spreading metric volume is $\Omega(C/\log n)$ (resp., $\Omega(C/\log T)$), as stated above. The crux of the argument is that the cost associated with an edge (or node) in a cutset can be bounded by the number of nodes between the previous and the next cutset in the sequence.

We point out that the methods in [3] applied to more problems, including the $d$-dimensional graph embedding problem and the minimum feedback arc set problem [15]. We could not extend our methods to these other problems, since we were unable to find a suitable bound on the cost of a sequence of cutsets associated with any of these problems.

Finally, for planar graphs and other undirected graphs that exclude some fixed minors, we combine a structural theorem of Klein, Plotkin, and Rao [8] with our new recursion techniques to show that the spreading metric cost volumes are within an $O(\log\log n)$ factor of the cost of the optimal solution for the MLA and the minimum containing interval graph problems.

**1.3. Overview.** In section 2, we present a formal definition of the MLA problem and define the spreading metric used for this problem. In section 3, we present a polynomial time $O(\log n)$-approximation algorithm for the MLA problem on an arbitrary graph with $n$ nodes and nonnegative edge weights. In sections 4 and 5, we define and briefly discuss the algorithms for approximating the minimum storage–time product problem and minimum containing interval graph problem, respectively. In section 6, we show how to improve the approximation factor for the MLA and the minimum containing interval problems to $O(\log\log n)$, in case the graph has no fixed $K_{r,r}$-minors—e.g., the graph is planar.

## 2. The MLA problem.

The *minimum linear arrangement* (MLA) *problem* is defined as follows: Given an undirected graph $G(V,E)$, with $n$ nodes, and nonnegative edge weights $w(e)$, for all $e$ in $E$, we would like to find a linear arrangement of the nodes $\sigma : V \rightarrow \{1,\dots,n\}$ that minimizes the sum, over all $(i,j) \in E$, of the weighted edge lengths $|\sigma(i)-\sigma(j)|$. In other words, we would like to minimize the *cost*

$$\sum_{(i,j)\in E} w(i,j)\,|\sigma(i)-\sigma(j)|$$

of a linear arrangement $\sigma$. In the context of VLSI layout, $|\sigma(i)-\sigma(j)|$ represents the length of the interconnection between $i$ and $j$.

We now define the spreading metric used in the algorithms for the MLA problem presented in sections 3 and 6. Analogous functions are used when approximating the minimum storage–time product problem (as presented in section 4) and the minimum containing interval graph problem (see section 5). Here we present the concept of spreading metrics in the context of the MLA problem (see [3] for a more general definition).

A *spreading metric* is a function $\ell : E \rightarrow Q$ that assigns rational lengths to every edge in $E$ and that can be computed in polynomial time. It also satisfies the two properties below. The *volume* of a spreading metric $\ell$ is given by $\sum_{e\in E} w(e)\ell(e)$.

1. *Diameter guarantee.* Let the distances be measured with respect to the lengths $\ell(e)$. The distances induced by the spreading metric "spread" the graph and all its nontrivial subgraphs. In this application, this translates to, "The diameter of every nontrivial connected subgraph $U$ of $V$ is $\Omega(|U|)$."

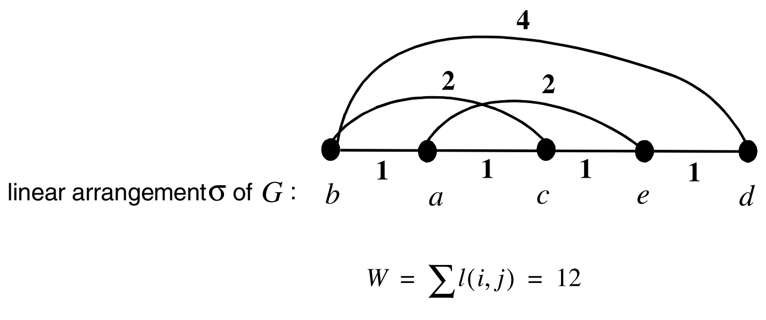


FIG. 2. *An assignment of lengths to the edges of G.*

2. *Lower bound.* The minimum volume of a spreading metric is a lower bound on the cost of an MLA of $G$.

A solution $\ell$ to (1)–(3) is a spreading metric for the MLA problem (see [3]). Let $\mathcal{V}$ denote the set of all nontrivial connected subgraphs of $V$.

$$W = \min_{e \in E} w(e)\ell'(e) \tag{1}$$

$$\text{s.t. } \frac{\left(\sum_{u \in U} \mathbf{dist}\ (u,v)\right)}{|U|} \geq \frac{|U|}{4} \quad \forall v \in U, \ \ \forall U \in \mathcal{V}, \tag{2}$$

$$\ell'(e) \geq 0 \qquad \forall e \in E, \tag{3}$$

where **dist** $(u,v)$ is the length of a shortest path from $u$ to $v$ according to the lengths $\ell'(e)$. The metric $\ell$ can be computed in polynomial time (see [3]) using, e.g., the ellipsoid method (there may be an exponential number of constraints in (2)). Note that (2) actually implies that $\ell(e) \geq 1$ for all $e$ in $E$—simply consider the subsets $U$ that consist of a single edge and its endpoints.

A solution $\ell$ to (1)–(3) is a *lower bound* on the cost of an MLA, since for any linear ordering $\sigma$ of the nodes of $G$, the assignment of lengths to the edges of $G$ given by $\ell'(i,j) = |\sigma(i) - \sigma(j)|$ satisfies (2)–(3). The volume of such an assignment is exactly the cost of $\sigma$. In particular this is true for an MLA $\sigma$. Hence $W = \sum_{e \in E} w(e)\ell(e)$ is less than or equal to the cost of an MLA. (Note that this lower bound is existentially tight, since there exist instances of this problem such that $\ell(i,j) = |\sigma(i) - \sigma(j)|$, where $\sigma$ is an MLA of $G$, as, for example, when $G$ is a linear array.) We will use this fact later, when proving Theorems 3.2 and 6.3. Figure 2 illustrates an assignment of lengths for the linear arrangement $\sigma$ given by the ordering of the nodes of $G$ from left to right (the lengths $\ell'(i,j)$ are the numbers associated with the edges in that picture; w.l.o.g.,[2] assume that all the edge weights are 1).

Let $\ell$ be a spreading metric of volume $W = \sum_{e \in E} w(e)\ell(e)$ that satisfies (1)–(3). In the remainder of this paper, all the distances in $G$ are measured with respect to $\ell$.

In [15], Seymour presents a lower bound (which he attributes to Noga Alon) on the gap between the volume of a spreading metric and an optimal integral solution for the minimum feedback arc set problem. In a nutshell, we can describe this lower bound when translated to the MLA problem as follows. Consider a bounded degree expander on $n$ nodes. An optimal solution of the spreading metric on this expander graph will assign a length of $O(n/\log n)$ to each edge (since, for any node $u$ in the

[2]Without loss of generality.

graph, there are roughly $n/2$ nodes at distance $\Theta(\log n)$ from $u$), incurring a volume of $O(n^2/\log n)$. Any integral solution to the minimum linear arrangement problem must stretch $\Omega(n)$ edges by $\Omega(n)$, leading to a lower bound of $\Omega(\log n)$ on the gap.

## 3. The algorithm.

We now present our $O(\log n)$-approximation algorithm for the MLA problem on general graphs. Let $G(V, E)$ be a graph with nonnegative edge weights $w(e)$. Assume w.l.o.g. that $G$ is connected (otherwise consider each connected component of $G$ separately), and that all the edge weights $w(e)$ are greater than or equal to 1.

In this paragraph, we introduce the notion of a *level* according to $\ell$. Fix a node $v$ in $V$. An edge $(x, y)$ belongs to *level* $i$ with respect to $v$ if and only if $\mathbf{dist}\ (v, x) \leq i$ and $\mathbf{dist}\ (v, y) > i$ for any $i \in N$. Note that an edge may belong to more than one level, and that there may be edges that do not belong to any level. Let the *weight of level* $i$, denoted by $\rho_i$, be the sum of the weights of the edges at level $i$.

We will partition the levels according to their weights. For ease of notation, we assume that $\log W$ is an integer (otherwise, simply use $\lceil \log W \rceil$ instead of $\log W$ below). We partition the levels into $\log W$ groups, according to the indices assigned to the levels. Let $\alpha_k = 2^k$ for all $k$ in $[(\log W) + 1]$.[3] Level $i$ has *index* $k$, $k$ in $[\log W]$, if and only if $\rho_i$ belongs to the interval $I_k = (\alpha_k, \alpha_{k+1}]$.

It follows from (2) that we must have at least $n/4$ distinct levels with nonzero weight. Note that since $w(e) \geq 1$ and $\ell(e) \geq 1$ for all $e$, any level with nonzero weight must have weight at least 1. Since there are $\log W$ distinct level indices, there must be at least $n/(4 \log W)$ levels with same index $k$, for some $k$. Let $\kappa$ be the exact number of levels of index $k$.

In a recursive step of the algorithm, we cut along the sequence of $\kappa$ levels of index $k$—i.e., we remove all of the edges that belong to at least one of these levels, even if they also belong to some other levels of an index different from $k$. A more detailed, stepwise description of the algorithm follows:

1. Select any node $v$ in the graph.
2. *Assign edges to levels.* An edge $(x, y)$ belongs to level $i$ with respect to $v$ if and only if $\mathbf{dist}\ (v, x) \leq i$ and $\mathbf{dist}\ (v, y) > i$ for any $i \in N$.
3. *Partition levels according to their indices.* Level $i$ has index $k$, $k$ in $[\log W]$, if and only if the weight of level $i$ (given by the sum of the weights of the edges at this level), $\rho_i$, belongs to the interval $I_k = (\alpha_k, \alpha_{k+1}]$, where $\alpha_k = 2^k$ for all $k$ in $[(\log W) + 1]$. Select an index $k$ such that there are $\kappa \geq n/(4 \log W)$ levels with this index.
4. *Cut along selected levels.* For all $i$, let level $a_i$ be the $i$th level of index $k$, in increasing order of distances to $v$. Let $H_i$ be the subgraph induced by the nodes that are at distance greater than $a_i$ and at most $a_{i+1}$ from $v$; let $H_0$ (resp., $H_\kappa$) be the subgraph induced by the nodes that are at distance at most $a_1$ (resp., greater than $a_\kappa$) from $v$. Let $n_i$ denote the number of nodes in $H_i$.
5. *Recursive step.* Recursively call the algorithm on each $H_i$, obtaining a linear arrangement $\sigma_i$ for the $n_i$ nodes in this subgraph.
6. *Combine.* Combine the linear arrangements obtained for the $H_i$'s, obtaining a linear arrangement $\sigma$ for $G$, as follows:

$$(\sigma(1), \ldots, \sigma(n)) = (\sigma_0(1), \ldots, \sigma_0(n_0), \sigma_1(1), \ldots, \sigma_1(n_1), \ldots, \sigma_\kappa(1), \ldots, \sigma_\kappa(n_\kappa)).$$

[3]For integer $x$ we use the notation $[x]$ to denote the set $\{0, \ldots, x - 1\}$.

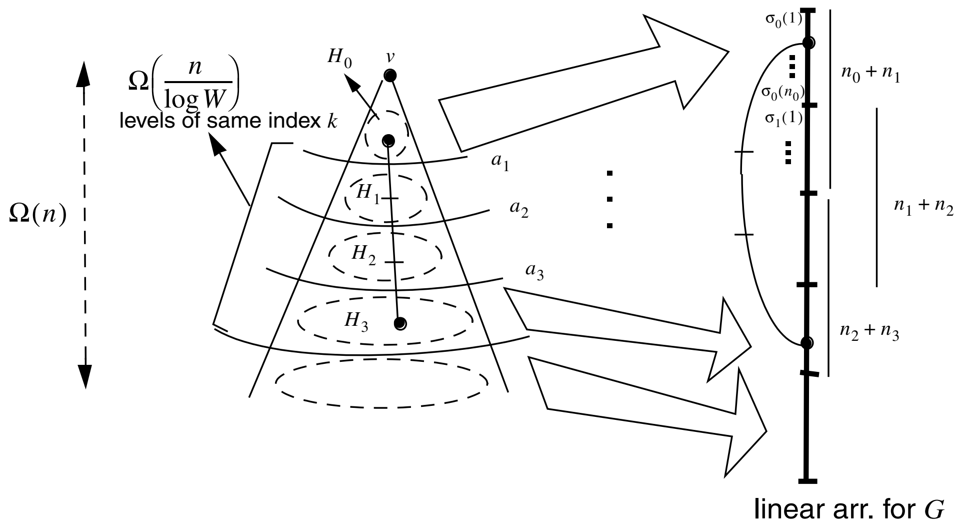


FIG. 3. *The algorithm and charging scheme.*

Each recursive step runs in polynomial time; at each recursive step, we decompose a connected component into at least two connected components. Hence the algorithm runs in polynomial time.

We use a *charging scheme* to account for the length of an edge $e$ in the linear arrangement for $G$ obtained by our algorithm (note that we will account for the *length of the edge in the linear arrangement*, rather than for the spreading metric length of the edge). If some edge $e$ in level $a_i$ belongs to some other level of index $k$, say level $a_j$, then this edge also belongs to every level of index $k$ between $a_i$ and $a_j$. W.l.o.g. assume that $i < j$. Edge $e$ will be "stretched over" all the nodes in $H_i \cup \cdots \cup H_{j-1}$ and may be "stretched over" some of the nodes in $H_{i-1}$ and $H_j$ in the linear arrangement produced by our algorithm. Hence the length of such an edge in the final linear arrangement will be at most $n_{i-1} + \cdots + n_j$. Suppose we charge $n_{p-1} + n_p$ for the portion of the edge that is stretched over the nodes in $H_{p-1} \cup H_p$, when considering level $a_p$, for all $i \leq p \leq j$. Then the total charge associated with edge $e$ is equal to $n_{i-1} + 2(n_i + \cdots + n_{j-1}) + n_j$—that is, edge $e$ will be charged at least as much as its length in a final linear arrangement.

Figure 3 illustrates the algorithm and charging scheme described above. On the left, we illustrate the selected levels $a_i$ along which we cut, resulting in the subgraphs $H_i$. After recursively calling the algorithm on these subgraphs, we obtain linear arrangements $\sigma_i$ for each $H_i$, which are concatenated—according to the distances from $v$ to each $H_i$—to form a linear arrangement of the original graph (illustrated on the right). The figure on the right also illustrates the charging scheme for one edge of the graph, which belongs to levels $a_1, a_2$, and $a_3$ in this example.

We will now compute an upper bound on the cost of a linear arrangement obtained by our algorithm. Let $C(Z)$ be the maximum cost of a linear arrangement obtained by our algorithm for a subgraph of $G$ whose volume of the spreading metric $\ell$ is at most $Z$. Since the sum of the weights of all edges in level $a_i$ is $\rho_{a_i}$, and since the quantity $w(e)\ell(e)$ for an edge $e$ which belongs to levels $a_i, \ldots, a_j$, $i \leq j$, satisfies $w(e)\ell(e) \geq w(e)(j - i + 1)/2$, we have that the sum of the weights of all edges that belong to some level of index $k$ is at least $\sum_{i=1}^{\kappa} \rho_{a_i}/2$. Note that this lower bound on the total sum of the weights of the edges removed in this cut step is tight: Suppose there are two consecutive levels $i$ and $i+1$ of index $k$, and suppose there is an edge $e$ which belongs to both of these levels such that $\ell(e) = 1 + \epsilon$ for some arbitrarily small $\epsilon > 0$; the quantity $w(e)\ell(e)$ is equal to $w(e)(1+\epsilon)$, which tends to $w(e)[(i+1) - i + 1]/2 = w(e)$ as $\epsilon$ tends to zero. We charge for the length of an edge as described in the preceding paragraph, and thus derive the following recurrence relation for $C(W)$:

$$C(W) \leq C\left(W - \sum_{i=1}^{\kappa} \frac{\rho_{a_i}}{2}\right) + \sum_{i=1}^{\kappa} [\rho_{a_i}(n_{i-1} + n_i)].$$

We now show that $C(W) = O(W \log n)$. We first prove the following lemma.[4]

LEMMA 3.1. *$C(W) \leq 32W \log(W+1)$.*

*Proof.* We will use induction on $W$. The base case $W = 0$ corresponds to a totally disconnected graph (a graph with no edges), and therefore $C(W) = 0$ in this case.

We can use induction on $W$ here since, for any subgraph of $G$ on $x$ nodes whose volume of the spreading metric $\ell$ is at most $Z$ ($Z \leq W$),

$$\sum_{i=1}^{\kappa} \frac{\rho_{a_i}}{2} \geq \frac{\alpha_k x}{8 \log Z} \geq \frac{\alpha_k x}{8 \log W} \geq \frac{1}{8 \log W}$$

since $\rho_{a_i} > \alpha_k$ and $\kappa \geq x/(4 \log W)$. Thus, the recurrence relation above will converge to the base case in at most $8W \log W$ steps.

Combining the recurrence relation for $C(W)$ with $\alpha_k < \rho_{a_i} \leq \alpha_{k+1}$ for all $i$, we obtain

$$\begin{aligned} C(W) &\leq C\left(W - \frac{\alpha_k n}{8 \log W}\right) + \alpha_{k+1} \sum_i (n_{i-1} + n_i) \\ &\leq 32 \left[W - \frac{\alpha_k n}{8 \log W}\right] \log\left[W - \frac{\alpha_k n}{8 \log W} + 1\right] + 2\alpha_{k+1} n \\ &\leq 32 \left[W - \frac{\alpha_k n}{8 \log W}\right] \log(W+1) + 2\alpha_{k+1} n \\ &\leq 32W \log(W+1) + \alpha_{k+1} n \left[2 - \frac{32}{16}\right] \\ &\leq 32W \log(W+1). \end{aligned}$$

The second inequality follows from the induction hypothesis; the fourth inequality follows since $\alpha_{k+1} = 2\alpha_k$. ☐

We still need to show how to bring the approximation factor down from $O(\log W)$ to $O(\log n)$. We will do this by using standard techniques of rescaling and rounding down the edge weights (as in [4]).

Our goal will be to reduce, by rescaling and rounding down weights, our original input graph $G$ to an "equivalent" input graph $G'$ whose spreading metric volume is a polynomial in $n$. Consider the set $E'$ of edges $e$ such that $w(e) \leq W/(mn)$. Since an edge has length at most $n$ in any linear arrangement for $G$, the contribution of the

[4] All the logarithms in this paper are base 2.

edges in $E'$ to an MLA of $G$ is at most $W$. Suppose we delete all those edges and apply a $\rho$-approximation algorithm to the resulting graph.

We now round down each weight $w(e)$, for all $e$ in $E \setminus E'$, to its nearest multiple of $W/(mn)$. The error incurred by this rounding procedure is again at most $W$. Furthermore, we scale the rounded weights by $W/(mn)$, obtaining new weights for the edges that are all integers in the interval $[0, mn]$. Note that we have only changed the units in which the weights are expressed.

The volume $W'$ of the spreading metric for solving the MLA problem on $G' = G \setminus E'$ with integral weights that belong to $[0, mn]$ is at most a polynomial on $n$ (since $W' \leq m^2n^2$). By Lemma 3.1, we have $C(W') \leq cW' \log(W') = c'W' \log n$ for some constant $c'$. Rescaling the edge weights back by multiplying $C(W')$ by $W/(mn)$, we obtain a linear arrangement for the original weights on $G'$ with cost at most $c'W \log n$. Putting back the edges in $E'$ into the linear arrangement obtained for $G'$, we obtain a linear arrangement for $G$ with cost at most $(c' \log n + 1)W$.

Since $W = w(e)\ell(e)$ is a lower bound on the cost of an MLA, by Lemma 3.1 and the considerations above, we have proved the following theorem.

THEOREM 3.2. *The cost of a solution to the MLA problem, obtained by our algorithm, is within an $O(\log n)$ factor of the cost of an MLA of $G$.*

## 4. Storage–time product.

In this section, we sketch our approach for approximating the minimum storage–time product problem on a directed acyclic graph $G(V, E)$. The minimum storage–time product problem arises in a manufacturing or computational process, in which the goal is to minimize the total storage–time product of the process: We want to minimize the use of storage over time, assuming storage is an expensive resource. Let $G(V, E)$ be an acyclic directed graph on $n$ nodes with edge weights $w(e)$ for all $e \in E$ and node weights $\tau(v)$ for all $v \in V$. The nodes of $G$ represent tasks to be scheduled on a single processor. The time required to process task $v$ is given by $\tau(v)$. The weight on edge $(u, v)$, $w(u, v)$ represents the number of units of storage required to save intermediate results generated by task $u$ until they are consumed at task $v$. The *minimum storage–time product problem* consists of finding a topological ordering[5] of the nodes $\sigma : V \to \{1, \ldots, n\}$ that minimizes

$$\sum_{(i,j)\in E, \sigma(i)<\sigma(j)} \left\{ w(i,j) \left[ \sum_{k \;:\; \sigma(i)\leq\sigma(k)\leq\sigma(j)} \tau(\sigma(k)) \right] \right\}.$$

Figure 4 illustrates a topological ordering of the nodes of $G$ (given from left to right on the rightmost representation of the graph) with minimum storage–time product of 177.

This problem generalizes the MLA problem: When all tasks have unit execution time, it becomes a directed version of the MLA problem. It is also a generalization of the single-processor scheduling problem, if we are minimizing the weighted sum of completion times (this problem is NP-complete [5, problem SS13, p. 240]).

For the storage–time product problem, we use a spreading metric $\ell$ defined as follows. Let $E^R = \{(u, v) | (v, u) \in E\}$. We define $G' = (V, E \cup E^R)$. Let $\mathcal{V}$ denote the set of all nontrivial strongly connected subgraphs of $G'$. The spreading metric $\ell$ is a

[5]An ordering $\sigma$ of the nodes of $G$ (where $G$ is an acyclic directed graph) is said to be *topological* if and only if for every $(i, j) \in E$, $\sigma(i) < \sigma(j)$.

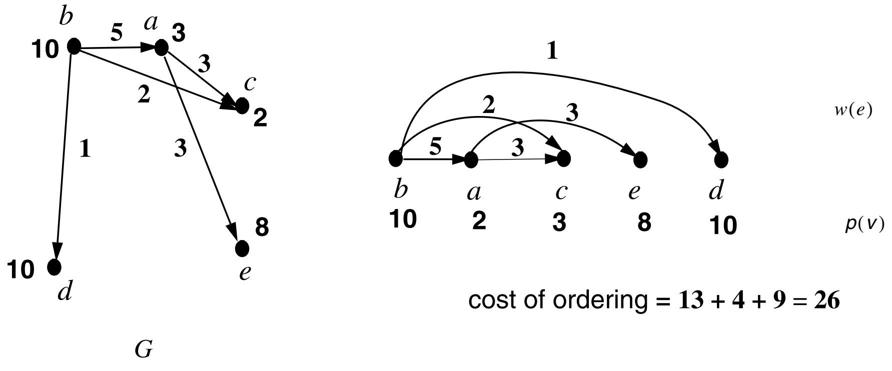


FIG. 4. *A minimum storage–time product of G.*

solution to

$$\begin{aligned} &W = \min w(e)\ell'(e) \\ \text{s.t.}\quad &\frac{\sum_{u\in U}\tau(u)\delta(u,v)}{|U|} \geq \frac{\sum_{u\in U}\tau(u)}{4} && \forall U \in \mathcal{V},\ \ \forall v \in U, \\ &\ell'(i,j) \geq \tau(i)+\tau(j) && \forall (i,j) \in E, \\ &\ell'(i,j) = 0 && \forall (i,j) \in E^R, \end{aligned}$$

where $\delta(u,v)$ is equal to $(\mathbf{dist}\ (u,v) + \mathbf{dist}\ (v,u))$. Here we define $\mathbf{dist}\ (u,v)$ to be the length of a shortest path from $u$ to $v$ in $G'$ according to the lengths $\ell'(e)$ for $e \in E \cup E^R$. (Note that each $e \in E^R$ has length zero.)

For any linear ordering $\sigma$ of $V$, the assignment of lengths to the edges given by $\ell'(i,j) = \sum_{k\ :\ \sigma(i)\leq\sigma(k)\leq\sigma(j)} \tau(\sigma(k))$, for all $(i,j)$ in $E$, satisfies the constraints above. Thus the volume $W$ of the spreading metric $\ell$ is a lower bound on the optimal cost of a solution to the storage–time product problem.

We can adapt the algorithm of section 3 to this problem as follows. Let $T = \sum_{v\in V}\tau(v)$. There is a node $v$ such that the out-tree rooted at $v$ or the in-tree rooted at $v$ has depth $\Omega(T)$. Thus, we can find a sequence $a_1,\ldots,a_\kappa$ of $\kappa = \Omega(T/\log W)$ levels whose weights $\rho_{a_1},\ldots,\rho_{a_\kappa}$ are within a factor of two of each other (as in section 3).

Laying out the resulting pieces successively, we obtain a solution where the cost is bounded by

$$C(W) \leq C\left(W - \sum_{i=1}^{\kappa}\frac{\rho_{a_i}}{2}\right) + \sum_{i=1}^{\kappa}[\rho_{a_i}(\tau_{i-1}+\tau_i)],$$

where $\tau_i$ is the sum of $\tau(v)$ over all nodes $v$ that lie between levels $a_{i-1}$ and $a_i$ ($\tau_0$ and $\tau_\kappa$ are defined accordingly).

This recursion can be upper bounded by $O(W\log W)$, as in section 3. This cost can be reduced to $O(W\log T)$ using the standard techniques that were used in section 3 to reduce $O(\log W)$ to $O(\log n)$.

**5. Minimum containing interval graph.** In this section, we sketch our approach to approximating the cost of a minimum containing interval graph of a graph $G(V,E)$.

We first introduce interval graphs. An *interval graph* is a graph whose nodes can be mapped to distinct intervals in the real line such that two nodes in the graph have an edge between them if and only if their corresponding intervals overlap. A completion of a graph $G$ into an interval graph results in an interval graph with the same node set as $G$ that contains $G$ as a subgraph.

We use the following characterization of interval graphs, due to [12]. An undirected graph $G(V, E)$ on $n$ nodes is an interval graph if and only if there exists a linear ordering $\sigma : V \to \{1, \ldots, n\}$ of the nodes in $V$ such that if an edge $(u, v) \in E$, where $\sigma(u) < \sigma(v)$, then every edge $(u, w)$, for $w$ such that $\sigma(u) < \sigma(w) < \sigma(v)$, also belongs to $E$. This characterization implies that, for any given ordering $\sigma$ of the nodes in $G$, there exists a *unique* way of completing $G$ into an interval graph $\overline{G}$ by adding as few edges to $G$ as possible.

The *minimum containing interval graph problem* seeks a completion of a graph $G$ into an interval graph $\overline{G}$ with *minimum total number of edges.* Thus, in this problem, the cost of a completed graph is given by the *total* number of edges in the graph. This cost can be viewed as the sum over nodes of the maximum backward stretch of the node—i.e., of the distance to the farthest lower-numbered node to which the node is connected. This is very similar to the MLA problem, except that the nodes are stretched along the order rather than the edges (see [3]). Thus, our techniques in sections 3 and 6 also apply to this problem. The containing interval graph problem arises in several areas, from computer science, to biology (see [11]), to archaeology (e.g., when finding a consistent chronological model for tool use while making as few assumptions as possible [7]).

The spreading metric $\ell$ that we use (due to [3]) assigns lengths to the nodes of the graph, rather than to its edges, as in the minimum linear arrangement and in the minimum storage–time product problems. Let $\mathcal{V}$ denote the set of all nontrivial connected subgraphs of $G$. The metric $\ell$ is a solution to

$$\begin{aligned} W = \min\ & \frac{1}{2}\left(\sum_{v \in V} \ell'(v)\right) \\ \text{s.t. } & \sum_{v \in U} \mathbf{dist}\,(u, v) \geq \frac{1}{4}(|U|^2 - 1) \qquad \forall u \in U,\ \forall U \in \mathcal{V}, \\ & \ell'(v) \geq 0 \qquad\qquad \forall v \in V, \end{aligned}$$

where $\mathbf{dist}\,(u, v)$ is the shortest length—given by $[\ell'(u) + \sum_{i=0}^{p} \ell'(x_i) + \ell'(v)]$—of a path $u, x_0, \ldots, x_p, v$, $x_i \in V$, from $u$ to $v$ in $G$.

Let $\overline{G}(V, \overline{E})$ be a completion of $G$ into an interval graph. If we let $\ell'(v)$ be the degree of node $v$ in $\overline{G}$, the cost $(\sum_{v \in V} \ell'(v))/2$ clearly gives the number of edges in $\overline{E}$. Also this assignment of lengths to the nodes satisfies the constraints above. Hence the volume $W$ of the metric $\ell$ is a lower bound on the number of edges in a minimum containing interval graph of $G$.

The recurrence relations that bound the cost of a solution obtained for the minimum containing interval graph problem are analogous to the ones for the MLA problem, both for the general case and for the excluded $K_{r,r}$-minors case (to be addressed in section 6).

A closely related problem to the minimum containing interval graph problem is that of the *minimum interval graph completion problem.* In this problem, the cost of a completion of a graph $G$ into an interval graph $\overline{G}$ is given by the number of edges in $\overline{G} - G$. While an optimal solution to the minimum containing interval graph problem on a graph $G$ is also an optimal solution to the minimum interval graph completion problem on $G$, an $\alpha$-approximate solution $\overline{G}$ to the minimum containing interval graph completion problem on $G$ is not enough to guarantee the existence of a solution to the minimum interval graph completion problem on $G$ with an $\alpha$-approximation factor.

**6. Graphs with excluded minors.** In this section we show how to obtain, in polynomial time, an $O(\log \log n)$-approximation for the MLA problem on a graph $G$ with no fixed $K_{r,r}$-minors—e.g., on a planar graph $G$. We denote the $r \times r$ complete bipartite graph by $K_{r,r}$.

DEFINITION 6.1. *Let $H$ and $G$ be graphs. Suppose that* (i) *$G$ contains disjoint connected subgraphs $A_v$ for each node $v$ of $H$; and* (ii) *for every edge $(u, v)$ in $H$, there is a path $P_{(u,v)}$ in $G$ with endpoints in $A_u$ and $A_v$, such that any node in $P_{(u,v)}$ other than its endpoints does not belong to any $A_w$, $w$ in $H$, nor to any $P_{(i,j)}$, $(i, j)$ in $H \setminus (u, v)$. Then $\cup_v A_v$ is said to be an $H$*-minor *of $G$.*

Klein, Plotkin, and Rao [8] showed how to decompose (in polynomial time) an undirected graph with no $K_{r,r}$-minors into connected components of small diameter. In our application, this implies that each connected component has at most a constant fraction of the nodes in $G$, as shown below.

The algorithm presented in this section also implies an $O(\log \log n)$-approximation for the minimum containing interval problem in excluded $K_{r,r}$-minor graphs, since this problem can be viewed as a "node version" of the MLA problem. Since the minimum storage–time problem is defined on a directed graph $G$, the decomposition techniques from Klein, Plotkin, and Rao [8] cannot be applied to $G$. Thus the results in this section do not translate into a better approximation factor for the minimum storage–time problem on graphs with no $K_{r,r}$-minors.

**6.1. The algorithm.** We recursively solve the problem, as we do in the general case. We combine the partial solutions returned by each recursive step, and we charge for each edge removed at a cut step in the same way as in the algorithm of section 3. It is in the way we decompose the graph before a recursive step (steps 2–4) that the algorithm of section 3 differs considerably from the one presented in this section. Before each recursive step, we will perform a series of *shortest path levelings*, to be defined soon, on each induced connected subgraph, until we can guarantee that the original graph has been decomposed into subgraphs that contain at most a fixed fraction (strictly less than one) of the nodes each. In the algorithm of section 3, we perform only one shortest path leveling before each recursive step.

The algorithm proceeds in rounds. In each round we have a *cut step*, which corresponds to the series of cuts performed during the round, and a *recursive step*, which consists of recursively calling the algorithm on the connected components that result from the cut step. Let $G(V, E)$ be a graph on $n$ nodes that excludes $K_{r,r}$ as a minor for some fixed $r > 0$. Let $\ell$ be a spreading metric for $G$ of volume $W$ that satisfies constraints (2)–(3).

A single *cut step* in $G$ will produce a series of subgraphs of $G$, $G = G_0, \ldots, G_t$, $t \leq r$, where each $G_{i+1}$ results from cutting according to a shortest path leveling of $G_i$. Fix a node $v$ in $G_i$. A *shortest path leveling* (SPL) of $G_i$ rooted at $v$ consists of an assignment of levels to the edges of $G_i$ as follows: An edge $(x, y)$ is at *level* $j$ of this SPL if and only if **dist** $(v, x)$ (in $G_i$) is at most $j$ and **dist** $(v, y)$ (in $G_i$) is greater than $j$, for all $j \in N$. (An edge may be at more than one level.)

We group the levels of this SPL into *bands* of $2s$ consecutive levels as follows: *Band* $i$, for $i \in N$, of the SPL consists of levels $2si$ through $2s(i + 1) - 1$, where $s = n/b$, and $b$ is a constant. Let $n(G_i)$ denote the number of nodes in $G_i$. The spreading metric diameter guarantee implies that this SPL has at least $n(G_i)/4$ levels. We will see later that $n(G_i) = \Theta(n)$, and that we can choose $b$ such that $n(G_i)/4 \geq 2s$ (we need $b \geq 8$). Alternate coloring the bands "blue" and "red" in increasing order of the levels.

We will cut along a sequence of levels of the SPL; one of the connected components resulting from this cutting procedure will be $G_{i+1}$. W.l.o.g., assume that the subgraph induced by the blue bands has at least $n(G_i)/2$ nodes. We have $2s$ cuts of the following type: For $0 \leq j \leq 2s-1$, a *leveled cut* $j$ consists of all the edges in the $j$th level (with respect to distance from $v$) of every red band. For example, if the band consisting of the first $2s$ levels is colored blue, then the leveled cut $j$ consists of the levels $2s+j, 6s+j, \dots$ for all $j$.

We group the leveled cuts according to their indices, which we define below (the definition of an index in this context is slightly different from that in section 3). We will see that there must exist at least an $\Omega(1/\log\log n)$ fraction of the leveled cuts with the same index $k_i$. Let $\beta_k = W2^k/(s\log n)$ for all $k$ in $[1, 2\log\log n]$. Let $\beta_0 = 0$ and $\beta_{(2\log\log n)} = W$. The *weight* of leveled cut $j$ is the sum of the weights of the levels in the cut (the weight of a level being the sum of the weights of the edges at that level). Leveled cut $j$ has *index* $k$, $k$ in $[2\log\log n]$, if and only if the weight of leveled cut $j$ belongs to the interval $I_k = (\beta_k, \beta_{k+1}]$. Thus there must exist at least $s/\log\log n$ leveled cuts in $G_i$ with same index $k_i$, since there are at least $2s$ distinct leveled cuts.

We summarize the steps above and conclude the description of the algorithm in the stepwise format below.

1. Let $i = 0$ and let $G_0 = G$.
2. Select a node $v$ in $G_i$.
3. *Assign edges to the leveled cuts of an SPL.* Assign the edges of $G_i$ to levels (we use the same definition of a level as in section 3), forming an SPL of $G_i$. We group the levels of the SPL into *bands* of $2s$ consecutive levels as described above. Alternate coloring the bands "blue" and "red" in increasing order of the levels. Assume, w.l.o.g., that the subgraph induced by the blue bands has at least $n(G_i)/2$ nodes. For $0 \leq j \leq 2s-1$, the $j$th *leveled cut* consists of all the edges in the $j$th level (with respect to distance from $v$) of every red band.
4. *Partition the leveled cuts of the SPL according to their indices.* Leveled cut $j$ has *index* $k$, $k$ in $[2\log\log n]$, if and only if the weight of leveled cut $j$ belongs to the interval $I_k = (\beta_k, \beta_{k+1}]$, where $\beta_k = W2^k/(s\log n)$ for all $k$ in $[1, 2\log\log n]$, $\beta_0 = 0$, and $\beta_{(2\log\log n)} = W$. Choose $k_i$ to be the index $k$ such that there exists at least $s/\log\log n$ leveled cuts in $G_i$ with same index $k$.
5. *Cut along selected leveled cuts.* If $i = r-1$, we let $t = i$ and go to step 6. If $i < r-1$, we proceed as described below.
   If $k_i > 0$, then we cut along these at least $s/\log\log n$ leveled cuts of index $k_i$ and weight at least $W/(s\log n)$. We recursively call the algorithm on the resulting connected components. In this case, we let $t = i$ and go to step 6.
   Otherwise, we first cut along only one of the leveled cuts of index $k_i = 0$ (chosen arbitrarily). Then we check whether there exists a resulting connected component $G_{i+1}$ of $G_i$ with at least $n(G_i)/2$ nodes. In case no such component exists, we let $t = i$ and go to step 6. If a component $G_{i+1}$ with at least $n(G_i)/2$ nodes exist, we proceed by going back to step 2 with $i = i+1$, thus performing an SPL on $G_{i+1}$.
6. *Recursive step.* The cut step is complete and we recursively call the algorithm on each of the resulting connected components $H_0, \dots, H_p$ of $G$. Let $n_i$ be

the number of nodes on component $H_i$, and let $\sigma_i$ be the linear arrangement obtained for $H_i$ for all $i$.

7. *Combine.* Assume the connected components $H_i$'s are ordered in nondecreasing order of distances from $v$ (i.e., $v$ is no closer to the nodes in $H_{i+1}$ than to the nodes in $H_i$). Combine the linear arrangements obtained for the $H_i$'s, obtaining a linear arrangement $\sigma$ for $G$, as follows:

$$(\sigma(1), \ldots, \sigma(n)) = (\sigma_0(1), \ldots, \sigma_0(n_0), \sigma_1(1), \ldots, \sigma_1(n_1), \ldots, \sigma_p(1), \ldots, \sigma_p(n_p)).$$

The number of nodes in $G_i$, $n(G_i)$, is proportional to $n$ for all $i$ in $[t]$. This follows since $n(G_{i+1}) \geq n(G_i)/2$, by the choice of $G_{i+1}$, and since $t$ $(\leq r)$ is a constant.

Suppose we just performed a series of $t$ SPLs and corresponding cut procedures. The last cut performed, on $G_{t-1}$, generated a collection of connected components of $G_{t-1}$. Klein, Plotkin, and Rao [8] proved that the distance in $G$ between any pair of nodes in any such component is $O(s)$ (where the constant in the $O(\cdot)$ notation depends only on $r$). Thus, for a large enough constant $b$, $b \geq 8$, we can ensure that the distance between any pair of nodes is at most $n/6$ in any such component.

We now show that it follows from the result by Klein, Plotkin, and Rao that any connected component that results from this cut step has at most $2n/3$ nodes. Fix any node $u$ in $G$. It follows from (2) that any subgraph of $G$ on $(n-x)$ nodes that contains $u$ has a node at distance at least $(n-x)/4$ from $u$. Suppose we start with the graph $G$ and proceed by removing one node at a time, choosing always a node that has maximum distance to $u$ among the remaining nodes. Thus, we need to remove at least one-third of the nodes before we are left only with nodes that are within distance $n/6$ from $u$ in $G$. This implies that any resulting connected component of $G_{t-1}$ has at most $2n/3$ nodes. Any other resulting connected component (of $G \setminus G_{t-1}$) has at most $n/2$ nodes, by the choice of the $G_i$'s.

We distinguish between two types of cut steps: If $k_{t-1}$ is equal to 0, then we have a cut step of *type I* in this round; otherwise $k_{t-1}$ is not zero, and the cut step in this round is of *type II*. Note that in either type of cut step, $k_j = 0$ for all $j$ in $[t-1]$.

Let $C(Z, x)$ denote the maximum cost of a linear arrangement obtained by our algorithm for a subgraph of $G$ with $x$ nodes, whose volume of the spreading metric $\ell$ is at most $Z$. Any graph with less than two nodes has no edges. Thus the cost of an MLA obtained by our algorithm and the volume of the corresponding spreading metric for a graph with $n < 2$ are both equal to zero. Hence it is sufficient to prove the claim below for $n \geq 2$ (note that $\log \log n$ is undefined for $n < 2$).

LEMMA 6.2. *$C(W, n) \leq cW \log \log n$ for $n \geq 2$ and some constant $c$.*

*Proof.* We use induction on $W$ and $n$ (a similar argument to that in Lemma 3.1 shows that induction converges to the base case here in a finite number of steps). The base cases for $W = 0$ or $n = 2$ are trivial. Suppose we perform a cut step of type I. Thus, we perform a sequence of at most $r$ cuts along a single leveled cut of each SPL in this step, and each of these leveled cuts has weight at most $2W/(s \log n)$. Let the connected components resulting from this cut step be $H_0, \ldots, H_p$. Then

$$\begin{aligned} C(W, n) &\leq \sum_{i=0}^{p} C(W_i, n_i) + r \frac{2W}{s \log n} n \\ &\leq \sum_i cW_i \log \log(2n/3) + \frac{2brW}{\log n} \\ &\leq cW \log \log n - \frac{cW}{3 \log n} + \frac{2brW}{\log n} \\ &\leq cW \log \log n, \end{aligned}$$

where $W_i$ and $n_i$ are the volume and number of nodes, respectively, associated with component $H_i$. We have shown that every $n_i$ is at most $2n/3$. The second inequality above follows by induction. Note that $\log\log(2n/3) \leq \log\log n - 1/(3\log n)$: Thus the third inequality follows. The last inequality follows for a sufficiently large constant $c$ ($c \geq 6br$).

If the cut step performed was of type II, then we performed a series of $t \leq r$ SPLs and respective cut procedures. The last term on the right-hand side of the first inequality below accounts for the first $(t-1)$th leveled cuts of index $k_i = 0$ used, one for each $G_i$, $0 \leq i < t-1$. The second term on the right-hand side of that inequality accounts for the $t$th group of leveled cuts of index $k_{t-1} > 0$ used for $G_{t-1}$. The charging scheme for the edges removed in the $t$th set of leveled cuts used in this cut step and the lower bound on how much those edges contributed to the spreading metric volume $W$ are analogous to the ones used in section 3.

$$\begin{aligned} C(W,n) &\leq C\left(W - \frac{\beta_k}{2}\frac{s}{\log\log n}, n\right) + 2\beta_{k+1}n \\ &\quad +(r-1)\frac{2W}{s\log n}n \\ &\leq C\left(W - \frac{\beta_k s}{2\log\log n}, n\right) + (r+3)\beta_k n \\ &\leq c\left(W - \frac{\beta_k s}{2\log\log n}\right)\log\log n + (r+3)\beta_k n \\ &\leq cW\log\log n + \beta_k n\left(r+3-\frac{c}{2b}\right) \\ &\leq cW\log\log n \end{aligned}$$

when $c \geq 2b(r+3)$. The second inequality above follows from $\beta_k \geq 2W/(s\log n)$, and from $\beta_{k+1} = 2\beta_k$, $0 < k < 2\log\log n - 1$; the fourth inequality follows since $s = n/b$. ∎

Since the volume $W$ of the spreading metric $\ell$ is a lower bound on the cost of an MLA, by Lemma 6.2, we obtain the following theorem.

THEOREM 6.3. *Given a graph $G$ on $n$ nodes that excludes fixed $K_{r,r}$-minors, the cost of a solution to the MLA problem obtained by the algorithm presented in this section is within an $O(\log\log n)$ factor of the cost of an MLA of $G$.*

## 7. Conclusion.

We provided an existentially tight bound on the relationship between the spreading metric cost volumes and the true optimal values for the problems of minimum linear arrangement, minimum containing interval graph, and minimum storage–time product.

It would be interesting to extend the techniques presented in this paper to obtain $O(\log n)$-approximation algorithms for other problems. In particular, it seems natural to extend our techniques to improve the best-known approximation factors for other problems that satisfy the "approximation paradigm" of [3], including the $d$-dimensional graph embedding problem and the minimum feedback arc set problem [15]. We would then provide an existentially tight bound—on the ratio between the value of an optimal solution and the spreading metric volume—for any such problem.

However, it is unclear whether the techniques presented in this paper could be directly extended to other problems in [3], since we heavily rely on the linear ordering properties of the solutions for the problems considered when finding a suitable bound on the cost of the sequence of cutsets used. It is unclear how to find a sequence of cutsets whose cost is $O(\log n)$ times the corresponding spreading metric volume due to the edges in the sequence of cutsets for the other problems in [3].

Another interesting problem would be to extend the ideas used for approximating the minimum containing interval graph problem to obtain better approximations for the minimum containing chordal graph problem (interval graphs are a particular subclass of chordal graphs). The minimum containing chordal graph problem has not been addressed in [3].

## REFERENCES

[1] C. F. Bornstein and S. Vempala, *Flow metrics*, in Proceedings of LATIN 2002: Theoretical Informatics, 5th Latin American Symposium, Lecture Notes in Comput. Sci. 2286, Springer-Verlag, Berlin, 2002, pp. 516–527.

[2] G. Even, J. Naor, S. Rao, and B. Schieber, *Fast approximate graph partitioning algorithms*, SIAM J. Comput., 28 (1999), pp. 2187–2214.

[3] G. Even, J. Naor, S. Rao, and B. Schieber, *Divide-and-conquer approximation algorithms via spreading metrics*, J. ACM, 47 (2000), pp. 585–616.

[4] G. Even, J. Naor, B. Schieber, and M. Sudan, *Approximating minimum feedback sets and multicuts in directed graphs*, Algorithmica, 20 (1998), pp. 151–174.

[5] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, W. H. Freeman, New York, 1979.

[6] M. Hansen, *Approximation algorithms for geometric embeddings in the plane with applications to parallel processing problems*, in Proceedings of the 30th Annual Symposium on Foundations of Computer Science, IEEE Computer Society Press, Los Alamitos, CA, 1989, pp. 604–609.

[7] D. G. Kendall, *Incidence matrices, interval graphs, and seriation in archeology*, Pacific J. Math., 28 (1969), pp. 565–570.

[8] P. Klein, S. Plotkin, and S. Rao, *Excluded minors, network decomposition, and multicommodity flow*, in Proceedings of the 25th Annual ACM Symposium on Theory of Computing, ACM, New York, 1993, pp. 682–690.

[9] R. Koch, T. Leighton, B. Maggs, S. Rao, and A. Rosenberg, *Work-preserving emulations of fixed-connection networks*, in Proceedings of the 21st Annual ACM Symposium on Theory of Computing, ACM, New York, 1989, pp. 227–240.

[10] F. T. Leighton and S. Rao, *Multicommodity max-flow min-cut theorems and their use in designing approximation algorithms*, J. ACM, 46 (1999), pp. 787–832.

[11] J. Meidanis and J. C. Setubal, *Introduction to Computational Molecular Biology*, PWS, Boston, MA, 1997.

[12] G. Ramalingam and C. P. Rangan, *A unified approach to domination problems in interval graphs*, Inform. Process. Lett., 27 (1988), pp. 271–274.

[13] S. Rao and A. W. Richa, *New approximation techniques for some ordering problems*, in Proceedings of the 9th Annual ACM-SIAM Symposium on Discrete Algorithms, ACM, New York, SIAM, Philadelphia, 1988, pp. 211–218.

[14] R. Ravi, A. Agrawal, and P. Klein, *Ordering problems approximated: Single-processor scheduling and interval graph completion*, in Proceedings of the 18th International Colloquium on Automata, Languages, and Programming, Lecture Notes in Comput. Sci. 510, Springer-Verlag, Berlin, 1991, pp. 751–762.

[15] P. D. Seymour, *Packing directed circuits fractionally*, Combinatorica, 15 (1995), pp. 281–288.

SIAM J. COMPUT.
Vol. 34, No. 2, pp. 405–432

# QUICK $k$-MEDIAN, $k$-CENTER, AND FACILITY LOCATION FOR SPARSE GRAPHS*

MIKKEL THORUP†

**Abstract.** We present $\tilde{O}(m)$ time and space constant factor approximation algorithms for the $k$-median, $k$-center, and facility location problems with assignment costs being shortest path distances in a weighted undirected graph with $n$ vertices and $m$ edges.

For all of these location problems, $\tilde{O}(n^2)$ time and space algorithms were already known, but here we are addressing large sparse graphs. An application could be placement of content distributing servers on the Internet. The Internet is large and changes so frequently that an $\tilde{O}(n^2)$ time solution would likely be outdated long before completion.

**Key words.** efficient approximation algorithms, shortest paths, location problems, $k$-median, $k$-center, facility location

**AMS subject classifications.** 05C12, 05C85, 68Q25, 68R10, 68W20, 68W25, 90C06, 90C27, 90C35, 90C59

**DOI.** 10.1137/S0097539701388884

## 1. Introduction.

We present efficient algorithms for several classical location problems defined in terms of a metric $(P, \text{dist})$, $|P| = n$. We want to pick a set $S \subseteq P$ of *facilities* subject to different objectives. By $\text{dist}(x, S)$ we mean the distance from $x$ to the nearest facility in $S$, that is, $\text{dist}(x, S) = \min_{f \in S} \text{dist}(x, f)$. In the *$k$-median* and *$k$-center problems* we require $|S| = k$, and our goal is to minimize $\sum_{x \in P} \text{dist}(x, S)$ and $\max_{x \in P} \text{dist}(x, S)$, respectively. In the *facility location problem*, there is no limit $k$ on the number of facilities, but we are further given a facility cost function f-cost, $P \rightarrow \mathbb{N}_0$, and our goal is to minimize $\sum_{a \in S} \text{f-cost}(a) + \sum_{x \in P} \text{dist}(x, S)$. Often one is really interested in an *augmentation version* of the problems, where one is given a set of existing facilities for free. For simplicity, we do not touch this issue, but it is straightforward to generalize the presented algorithms to deal with such augmentations.

In this paper, we are interested in the *graph setting* where the metric is the shortest path metric of a weighted undirected graph $G = (V, E, \ell : E \rightarrow \mathbb{N})$, $|V| = n$, $|E| = m$; that is, $\text{dist}(x, y)$ is the length of the shortest path from $x$ to $y$ in $G$. This setting is the basis for many of the classical applications of facility location. For example, a 1983 survey [26] describes applications of the $k$-median and $k$-center problems in mostly sparse networks using shortest path distances as cost metric (as in this paper) and mentions about 100 related references. A typical example is the placement of shopping centers on a road network with driving distance to the nearest shopping center being the consumer cost. A more contemporary example could be the placement of content distribution servers on the Internet (see [20, 21, 25] for more details on Internet-related applications). Both examples may concern large sparse graphs. Also, in both cases $k$ may be large; e.g., McDonald's has more than 28,000 fast-food restaurants on the US road network which has millions of road intersections.

*Received by the editors May 3, 2001; accepted for publication (in revised form) August 24, 2004; published electronically February 3, 2005. A short preliminary version of this work was presented at the 28th International Colloquium on Automata, Languages and Programming (ICALP'01); see [27].
http://www.siam.org/journals/sicomp/34-2/38888.html

†AT&T Labs—Research, 180 Park Avenue, Florham Park, NJ 07932 (mthorup@research.att.com).

Similarly, Akamai has thousands of servers on the Internet which is heading towards a million routers. For Internet-related applications, we further note that the Internet changes frequently, and thus an algorithm has to be fast in order to produce up-to-date answers.

For all of the above location problems, we present $\tilde{O}(m)$ time and space constant factor approximation algorithms. Here $\tilde{}$ means that we ignore log-factors, including a log in the maximal edge weight. The concrete approximation factors are $12 + o(1)$ for $k$-median, 2 for $k$-center, and $3 + o(1)$ for facility location. The approximation factor for the $k$-median problem may be reduced to around 9, but this is complicated and beyond the scope of the current paper. Also, using much more complicated techniques, the author has recently shown that the approximation factor for facility location can be reduced to 1.62 while preserving the near-linear time and space [28]. We note that a constant factor approximation is the best we can hope for in polynomial time for each of these location problems. More precisely, for $k$-median a factor of 1.36 is the best possible unless $NP = P$ [10]; for $k$-center, a factor of 2 is the best possible unless $NP = P$ [14]; and for facility location, a factor of 1.46 is the best possible unless $NP \subseteq DTIME(n^{\log \log n})$ [11]. Our focus here, however, is on reasonable running times rather than on the concrete constant in the approximation factor.

Most previous theoretical work on the above location problems has been focused on finding the best approximation factors in polynomial time (see, e.g., [2] and references therein). A notable exception is the work of Jain and Vazirani [19]. They considered a *distance oracle* setting where, given any pair of points $(x, y) \in P^2$, one can compute $\text{dist}(x, y)$ in constant time. The distance oracle setting is interesting in its own right, e.g., interpreting the location problems as clustering problems for large data bases [12]. For the $k$-median and facility location problems, Jain and Vazirani achieved approximation factors of 6 and 3, respectively, in $\tilde{O}(n^2)$ time,[1] improving for the $k$-median the LP-based factor $6\frac{2}{3}$ from [2]. They noted, "The distinguishing feature of our algorithms is their low running time." In their final discussion they ask whether improved running times can be obtained for the graph setting in the case of sparse graphs. To run their distance oracle algorithm on a graph, one would first have to compute all pairs of shortest paths in $\tilde{O}(mn)$ time. (Some other alternatives will be discussed in section 1.3.) Our positive answer to Jain and Vazirani's question is that constant factor approximations can be achieved with high probablility (w.h.p.) in $\tilde{O}(m)$ time.

The central result of this paper is the near-linear time solution to the $k$-median problem, the difficulty being the sharp bound on the number of facilities. Facility location is comparatively easy because we can use approximate facility costs; it is considered here for completeness because it was part of Jain and Vazirani's open problem. The $k$-center solution comes in for free as a warm-up for our solution to the facility location problem. Also, by covering $k$-median, $k$-center, and facility location, we develop a quite general toolbox for basic location problems in networks [26].

**1.1. Efficient algorithms for NP-hard problems.** For this paper, we have adopted the standard target in the algorithms field of minimizing the asymptotic time and space efficiency [6]. We make this point because within approximation algorithms there has been a tendency towards putting more emphasis on whether a polynomial time algorithm is "combinatorial," "primal-dual," "greedy," or "local search."

[1]Actually, they get $\tilde{O}(fn)$ time if only $f$ points are potential facilities, but here we generally assume that all points are potential facilities.

Obviously efficient algorithms are as important for NP-hard problems as for problems in P, especially since most important combinatorial optimization problems are NP-hard [7]. Of course, the design of efficient algorithms for NP-hard problems has to await an understanding of approximability of such problems in polynomial time. Fortunately, this understanding is emerging for many NP-hard optimization problems, thus opening many new challenges for the design of efficient algorithms. As the design of efficient approximation algorithms for NP-hard problems blossoms, it is expected that many new ideas will emerge due to the difference in nature between NP-hard problems and problems in P. In particular, it is very natural to trade approximation for efficiency, as approximation has already been traded for polynomial time. Some recent examples of this line of research may be found in [5, 15]. One way of thinking about the trade-off between time and approximation factor is that one should go for the best approximation factor that can be obtained within the time available. Thus, if one has $O(mn)$ time available for solving the $k$-median problem in a network, Jain and Vazirani's algorithm is superior to ours. However, with $o(mn)$ time available, our $\tilde{O}(m)$ time solution is the right choice.

To appreciate our improved running time, consider placing shopping centers on a U.S. map with several million intersections/vertices. Computing all pairwise shortest paths for a distance oracle algorithm would take several months, if not years. Of course, one might be able to reduce the problem by hand, but our goal here is to construct algorithms not relying on human preprocessing. An even worse problem would be the placement of servers on the full Internet graph, which is changing so rapidly that the input graph would be outdated long before an all-pairs shortest path computation finished. For contrast, our algorithm runs in time close to that of a single source shortest path algorithm. In addition to speed problems, even if we don't count the space needed to represent the pair-wise distances, it is not obvious how to implement Jain and Vazirani's algorithm in subquadratic space, unless we substantially increase the running time. Our algorithm needs only $\tilde{O}(m)$ space, which could be crucial in avoiding external memory problems. We note that we cannot expect future faster computers to solve our problems, for as computers grow faster, the problems grow bigger, and hence so does the need for near-linear time algorithms.

**1.2. The distance oracle version.** Our work was strongly inspired by Indyk's [15] work on finding "sublinear" $o(n^2)$ time algorithms in the distance oracle setting, saving time by only querying $o(n^2)$ distances. Our basic question was whether similar savings were possible in the graph setting, where individual distance queries take linear time.

For the $k$-median problem, Indyk [15] presented a randomized reduction that, together with the $\tilde{O}(fn)$ time factor 6 approximation algorithm of Jain and Vazirani [19], implies a $\tilde{O}(k^3n)$ time factor $(3 + o(1))(2 + 6) = 24 + o(1)$ bicriteria approximation using $2k$ facilities. Thus Indyk's solution may use up to $2k$ facilities, and it has a cost that is at most a factor $24 + o(1)$ from the optimal solution using only $k$ facilities. Later, based on Indyk's construction [15], Guha et al. [12] presented a $\tilde{O}(kn)$ time factor $6 \times 2 \times (24 + o(1) + 1) = 300 + o(1)$ approximation algorithm using only $k$ facilities, as required for the $k$-median problem. Their algorithm works for a special streaming version of the problem that we shall return to later in section 5.3. Their approximation factor can be reduced to $80 + o(1)$ if $k = \tilde{O}(\sqrt{n})$ [Guha, personal communication]. An $\tilde{O}(kn)$ time constant factor approximation is also announced by Mettu and Plaxton [23], but with no specification of the constant. They also pointed out that $\Omega(kn)$ time is necessary for any approximation guarantee even for randomized algorithms.

Some of our initial developments actually imply an $\tilde{O}(kn)$ time factor $12 + o(1)$ approximation for the distance oracle version of the $k$-median problem, thus doing 25 times better than the previous $\tilde{O}(kn)$ time algorithm for unbounded $k$. Clearly some of the previous approximation factors could have been improved somewhat, but our vast improvement is due to some new simple and powerful sampling techniques.

For the $k$-center algorithm, the standard factor 2 approximation [9, 13] uses only $\tilde{O}(kn)$ distance queries, and $\Omega(kn)$ queries are needed for any approximation guarantee. For facility location, $\Omega(n^2)$ distance queries are needed, and that is matched by Mettu and Plaxton's $O(n^2)$ time factor 3 approximation [23].

**1.3. The graph version.** The problem in the graph setting is that a single distance query is answered by a single source shortest path computation, computing the distance from a source to all other points. Thus, answering just $n$ distance queries independently is as slow computing all pairwise shortest paths.

Nevertheless, the previous work on the distance oracle version does have some applications for the graph version. Taking the $k$-median problem as the main example, first we can apply the all pairs small-stretch path algorithm of Cohen and Zwick [4]. In $\tilde{O}(n^2)$ time, this gives us all distances in the graph within a factor 3. Next, we can apply Jain and Vazirani's algorithm [19] in $\tilde{O}(n^2)$ time to get a factor $3 \times 6 = 18$ approximation algorithm. In order to exploit the $\tilde{O}(kn)$ time distance oracle algorithm of Guha et al. [12], we can first apply Thorup and Zwick's [29] approximate distance oracle algorithm: for any positive integer $t$, after $O(tmn^{1/t})$ preprocessing time, we can answer distance queries within a factor $2t - 1$ in $O(t)$ time. Setting $t = 2$ and combining it with [12], we get an $\tilde{O}(m\sqrt{n} + nk)$ time algorithm with approximation factor $3(300 + o(1)) = 900 + o(1)$, or $3(80 + o(1)) = 240 + o(1)$ if $k = O(\sqrt{n})$.

Our $\tilde{O}(m)$ time bound is near-optimal and breaks the $\Omega(kn)$ lower bound for distance oracles if $k \gg m/n$. Further, our approximation factor is only $12 + o(1)$, which is better than that of any previous $o(nm)$ algorithm.

Our algorithm is easily generalized to work for weighted points. One application of this arises if only a subset of the points are potential facilities. Then we first assign each point to its nearest potential facility. Second we solve the $k$-median problem over the potential facilities, each weighted by the number of assigned original points. The resulting solution can be seen to be a factor $2(12+o(1))+1 = 25+o(1)$ approximation.

**1.4. All points nearest marked neighbor.** What makes fast location algorithms possible for graphs is that we can quickly find the nearest facility of each point. More abstractly, the *all points nearest marked neighbor problem* concerns a metric $(P, \text{dist})$ with a set $S \subseteq P$ of marked points. The task is for each $x \in P$ to find a nearest marked point $x^S \in S$ as well as its distance $\text{dist}(x, x^S) = \text{dist}(x, S)$ from $x$. If $S$ is a set of facilities, having solved the all points nearest marked neighbor problem, we can easily compute the $k$-median, $k$-center, or facility location cost of $S$ in $O(n)$ time.

With a distance oracle, we need $\Theta(n|S|)$ time to find all points nearest neighbor in $S$. However, when $P$ is the vertex set of a graph and dist the shortest path distances, we can find all points nearest neighbor in $S$ in near-linear time with a single source shortest path computation, no matter the size of $S$. More precisely, we have the following.

Observation 1. *In a weighted connected graph with $m$ edges, we can solve the all points nearest marked neighbor problem in $\tilde{O}(m)$ time.*

*Proof.* Introduce a source $s$ with zero length edges to all $a \in S$, and compute the shortest path tree to all vertices in $\tilde{O}(m)$ time. For each $x$, $\mathrm{dist}(x, S)$ is the found distance from $s$. Also, for each $x$ in the subtree of $a \in S$, we set $x^S = a$. We note that the same construction works for directed graphs if we first reverse the direction of all edges. ∎

All our results are achieved by a polylogarithmic number of all points nearest marked neighbor computations, hence by a polylogarithmic number of single source shortest paths computations.

Our solution to the $k$-median problem via a small number of all points nearest marked neighbor computations is also useful in Hamming space, where it gives a constant factor approximation in $\tilde{O}(nk^{\varepsilon})$ time for $\varepsilon > 0$, thus beating that $\Omega(kn)$ lower bound for general distance oracles if $k = \log^{\omega(1)} n$.

**1.5. Contents.** After some preliminaries in section 2, we present our solutions to the $k$-center, facility location, and $k$-median problem in sections 3, 4, and 5. Our solution to the $k$-median problem is by far the hardest result in the paper, taking up most of the space.

## 2. Preliminaries.

Unless otherwise stated, logarithms are base 2; e.g., $\log n = \log_2 n$.

When we say *w.h.p.*, we mean with probability $\geq 1 - 1/n^{\omega(1)}$. The advantage of an error probability of $1/n^{\omega(1)}$ is that even if we multiply it by $n$, it remains $1/n^{\omega(1)}$.

Let $S$ be a set of facilities. We will generally use $x^S$ to denote the facility that a point $x$ is assigned to. Typically $x^S$ is the nearest point in $S$ as found by an all points nearest marked neighbor computation. By the $S$-cluster of $a \in S$ we mean the set $\{x \in P : x^S = a\}$, denoted $\mathrm{cluster}(a, S)$.

Generally, we will use a subscript to indicate that measurements are done in a metric different from the one currently understood. For example, if $H$ is a graph, $\mathrm{dist}_H(x, y)$ is the distance from $x$ to $y$ in $H$.

In our stated running times for graphs, we will assume that there are no isolated vertices, and hence that $m + n = O(m)$. Isolated vertices are trivially dealt with, in that each of them has to be a facility; otherwise the cost is $\infty$.

The results presented here are not intended to be of immediate practical value. The focus is on running times and approximation factors for arbitrary input instances, and for the running times we will be sloppy with log-factors in order to facilitate the simplest possible analysis.

## 3. $k$-center.

We want to pick $S \subseteq V$, $|S| = k$ minimizing $\max_{v \in V} \mathrm{dist}(v, S)$. A factor 2 approximation is classical [9, 13] and best possible [14], but the natural algorithm takes $\tilde{O}(km)$ time. We get down to $\tilde{O}(m)$ expected time, and our methods will be reused for facility location. Here, for the $k$-center problem, the $\tilde{O}$-notation includes a log-factor in the maximal edge weight.

The classical factor 2 approximation is the following greedy algorithm: Guess the optimal distance $d^*$, and then return any maximal dist-$2d^*$ independent set. Here, for any $d$, a subset $U \subseteq V$ is dist-$d$ *independent* if no two vertices of $U$ are within distance $d$ of each other. We know we have an adequate value of $d$ if it gives rise to $\leq k$ facilities whereas $d - 1$ gives rise to $> k$ facilities, so $d$ may be found with a binary search. If $N$ is the maximal edge weight, the maximal distance is less than $nN$, so the depth of the binary search is less than $\log_2 nN$.

The obvious greedy way of finding a maximal dist-$d$ independent set $U$ is as follows: set $U = \emptyset$ and $W = V$. While $W \neq \emptyset$, add an arbitrary vertex $v \in W$ to $U$

and remove all vertices within distance $d$ from $W$. The complexity of this algorithm is $\tilde{O}(|U|m)$. Here we show how to find a maximal dist-$d$ independent set in near-linear time. Including the binary search for $d$, this will give us a near-linear time solution to the $k$-center problem.

Let $\Gamma_{\leq d}(v)$ denote the neighborhood of vertices within distance $d$ of $v$. We shall use the following result of Cohen [3].

LEMMA 2 (Cohen). *Let $\varepsilon > 0$. For any $W \subseteq V$, after preprocessing in $\tilde{O}(m/\varepsilon)$ time and space, w.h.p, for every vertex $v$ and $d$, in $O(\log\log n - \log\varepsilon)$ time, we can estimate the size of $\Gamma_{\leq d}(v) \cap W$ within a factor $1 \pm \varepsilon$.*

In applications, we will typically use $\varepsilon = 1/\log n$ so that $\tilde{O}(m/\varepsilon) = \tilde{O}(m)$, $O(\log\log n - \log\varepsilon) = O(\log\log n)$, and $1 \pm \varepsilon = 1 \pm o(1)$. We now have the following maximal dist-$d$ independent set algorithm.

ALGORITHM A. Finds a maximal dist-$d$ independent set $U \subseteq W$ for a given $W \subseteq V$.

```
A.1.     set $U = \emptyset$
A.2.     while $W \neq \emptyset$,
A.2.1.       using Lemma 2, compute $\delta = (1 \pm o(1)) \max_{v \in W} |\Gamma_{\leq d}(v) \cap W|$
A.2.2.       pick a random subset $R$ of $W$ of size $|W|/\delta$
A.2.3.       let $T = \{v \in R | \Gamma_{\leq d}(v) \cap R = \{v\}\}$—see implementation comments below
A.2.4.       add $T$ to $U$
A.2.5.       using Observation 1, find dist$(v, U)$ for each $v \in W$, removing those with
             dist$(v, U) \leq d$
```

To identify the set $T$ in step A.2.3, we construct a family $\{R_i\}_{i \leq 2\log_2 |R|}$ of subsets of $R$ such that for all $v, w \in R$ there is a set $R_i$ containing $v$ but not $w$. These sets may be constructed by associating different $\log_2 |R|$ bit vectors to the vertices in $R$ and then characterizing each set by the value of a certain bit position. Now, $\Gamma_{\leq d}(v) \cap R = \{v\}$ if and only if dist$(v, R_i) > d$ for each $R_i$ not containing $v$. For each $R_i$, using Observation 1, we find dist$(v, R_i)$ for all $v$ in near-linear time. Hence, we can implement step A.2.3 in near-linear time.

PROPOSITION 3. *Algorithm* A *finds a maximal* dist-$d$ *independent subset of a given set $W \subseteq V$ in $\tilde{O}(m)$ expected time.*

*Proof.* It follows directly from the construction that we get a maximal dist-$d$ independent subset $U \subseteq W$. Also, it is clear that each iteration of the while-loop takes near-linear time. We want to show that we expect

$$\delta^* = \max_{v \in W} |\Gamma_{\leq d}(v) \cap W|$$

to be reduced by a constant factor within $O(\log n)$ iterations. This will imply that the total expected number of iterations is $O(\log^2 n)$.

Consider a specific iteration. We assume that $\delta = (1 \pm o(1))\delta^*$, as is the case w.h.p. Consider any vertex $v \in W$ with $|\Gamma_{\leq d}(v) \cap W| \geq 0.6\,\delta^* \geq \delta/2$. We will show that the iteration eliminates $v$ from $W$ with constant probability. The condition $|\Gamma_{\leq d}(v) \cap W| \geq \delta/2$ implies that some vertex $u \in \Gamma_{\leq d}(v) \cap W$ is picked for $R$ with constant probability. On the other hand, $|\Gamma_{\leq d}(u) \cap W| \leq \delta^* \leq (1 + o(1))\delta$, and thus with constant probability no other vertex from $\Gamma_{\leq d}(u)$ is picked for $R$. By definition, this means that $u$ ends in $T$, and since $u \in \Gamma_{\leq d}(v)$, this eliminates $v$ from $W$. Thus we conclude that if a vertex $v$ has $|\Gamma_{\leq d}(v) \cap W| \geq 0.6\,\delta^*$, then the iteration eliminates $v$ with constant probability.

Consider a specific value of $\delta^*$. There are at most $n$ vertices $v$ with $|\Gamma_{\leq d}(v) \cap W| \geq 0.6\,\delta^*$, and each such $v$ is eliminated with constant probability in each of the following iterations. Thus, we expect all of them to be eliminated within $O(\log n)$ iterations, thereby reducing $\delta^*$ by a factor 0.6. We can have only $O(\log n)$ such reductions of $\delta^*$, and so we conclude that the total expected number of iterations is $O(\log^2 n)$, and hence that the total expected running time of Algorithm A is near-linear. ∎

We note that the above proof has some similarities to the randomized parallel independent set algorithm in [22]. However, the algorithm in [22] accesses all edges, whereas we do not want to consider the $O(n^2)$ pairs of distance $\leq d$.

THEOREM 4. *In $\tilde{O}(m)$ expected time, we can find a factor 2 approximation to the $k$-center problem.*

*Proof.* As discussed earlier, we make a binary search over $\{1, \dots, nN\}$ for a value of $d$ such that the set $U_d$ returned by Algorithm A has at most $k$ elements, whereas the set $U_{d-1}$ returned for $d-1$ has more than $k$ elements. Then $U_d$ is our factor 2 approximate solution to the $k$-center problem. ∎

In [13] several similar problems are mentioned that can be improved with similar methods.

We note that the above algorithms, including those of Cohen [3], are based on a polylogarithmic number of all points nearest marked neighbor computations plus an $\tilde{O}(n)$ time overhead.

## 4. Facility location.

In the *facility location problem* for a metric $(P, \text{dist})$, there is a facility cost f-cost$(x)$ associated with each $x \in P$, and then the cost of $S \subseteq P$ is $\sum_{f \in S} \text{f-cost}(f) + \sum_{x \in P} \text{dist}(x, S)$. First we note that in the distance oracle version of the facility location problem, even if all facility costs are uniform, no constant factor approximation is possible with $o(n^2)$ queries, not even if we allow randomization. For a negative example, divide into clusters of size $t$ with intra- and inter-cluster distance 0 and $\infty$, respectively. Then we expect to need $\Omega(n^2/t)$ queries for an approximation factor substantially below $t$. It follows that Mettu and Plaxton's [23] $O(n^2)$ time bound is optimal for facility location with distance oracles, even with uniform facility costs.

However, in [8] (see also [16, p. 73]), it is shown that Jain and Vazirani's facility location algorithm [19] can be implemented with $\tilde{O}(n)$ nearest neighbor queries, leading to a more efficient solution in Hamming space [17]. We note that [8] does not give anything for the $k$-median problem, as it is based on approximate counting and hence approximate payment of facilities, and then Jain and Vazirani's rounding trick [19, section 3.2] from facility location to $k$-medians does not work.

In a graph, we have no efficient way of supporting individual nearest neighbor queries, but we can solve them efficiently for all points together as the all points nearest marked neighbor problem. Essentially, we show below that facility location can be solved within a factor 3 from optimality with a polylogarithmic number of solutions to the all points nearest neighbor problem. We note that whereas "phase 2" of Jain and Vazirani's algorithm [19] is trivial to implement with efficient individual nearest neighbor queries [8], it needs something like Proposition 3 for graphs.

Instead of using Jain and Vazirani's algorithm [19], we use the one of Mettu and Plaxton [23]. The factor 3 approximation algorithm of Mettu and Plaxton for facility location is very simple and elegant. As for the algorithm in [19], it has two phases.

*Phase* 1. For each $x \in P$, we find $r_x$ such that value$(x, r_x) =$ f-cost$(x)$, where value$(x, r) = \sum_{y \in P, \text{dist}(x,y) \leq r} (r - \text{dist}(x, y))$.

*Phase* 2. Starting with $S = \emptyset$, we visit $x \in P$ in order of increasing $r_x$, adding $x$ to $S$ if $\text{dist}(x, S) > 2r_x$.

Lemma 5 (see Mettu and Plaxton [23]). *The above set $S$ is at most a factor 3 from the optimal solution to the facility location problem.*

For an efficient implementation of the above algorithm, let $\varepsilon = \Theta(1/\log n)$ be such that 2 is an integral power of $(1+\varepsilon)$ for some integer $i$. Increasing assignment costs just a little, we will view all distances as rounded up to the nearest integral power of $(1+\varepsilon)$. We use $\text{dist}^{\uparrow}$ to denote this distance function, and let $\text{value}^{\uparrow}$ denote the corresponding change in value, that is, $\text{value}^{\uparrow}(x,r) = \sum_{y\in P, \text{dist}^{\uparrow}(x,y)\le r}(r - \text{dist}^{\uparrow}(x,y))$.

Note that when we use $r_x$ in Phase 2, all we care about is which vertices are within distance $2r_x$ from $x$. Let $i$ be such that $(1+\varepsilon)^i \le r_x < (1+\varepsilon)^{i+1}$. Since 2 is an integral power of $(1+\varepsilon)$, there are no rounded distances between $2(1+\varepsilon)^i$ and $2(1+\varepsilon)^{i+1}$. Thus, we can round $r_x$ down to the nearest power of $(1+\varepsilon)$, even if this implies $\text{value}^{\uparrow}(x, r_x) \ll \text{f-cost}(x)$. The rounded down value of $r_x$ is denoted $r_x^{\downarrow}$.

Algorithm B. Implements an approximate version of Phase 1, finding an $r_v^{\downarrow}$ for each $v \in V$.

```
B.1.        for all $v \in V$, set $f_v = 0$
B.2.        set $U = V$
B.3.        for $i = 1, 2, \dots$ while $U \neq \emptyset$,
B.3.1.          using Lemma 2 with $W = V$, estimate for each $u \in U$ the number $q_u$ of
                vertices $v$ within distance $(1+\varepsilon)^i$ from $u$, that is, $\text{dist}(u,v) \le (1+\varepsilon)^i$,
                or equivalently, $\text{dist}^{\uparrow}(u,v) \le (1+\varepsilon)^i$.
B.3.2.          for each $u \in U$,
B.3.2.1.            $f_u = f_u + ((1+\varepsilon)^i - (1+\varepsilon)^{i-1})q_u$
B.3.2.2.            if $f_u > \text{f-cost}(u)$,
B.3.2.2.1.              set $r_u^{\downarrow} = (1+\varepsilon)^{i-1}$
B.3.2.2.2.              remove $u$ from $U$
```

The following lemma states that Algorithm B performs an approximate implementation of Phase 1.

Lemma 6. *For each $v$, there is an $\text{f-cost}^{\approx}(v) = \text{f-cost}(v)/(1\pm\varepsilon)$ and an $r_v^{\approx}$ with $\text{value}^{\uparrow}(v, r_v^{\approx}) = \text{f-cost}^{\approx}(v)$ such that $r_v^{\downarrow}$ is $r_v^{\approx}$ rounded down to the nearest power of $(1+\varepsilon)$.*

*Proof.* First, suppose that the numbers $q_u$ counted the number of vertices within distance $(1+\varepsilon)^i$ exactly. Then, after step B.3.2.1, we would have $f_u = \text{value}^{\uparrow}(v, (1+\varepsilon)^i)$. However, each $q_u$ may be off by a factor $(1\pm\varepsilon)$, and hence so is $((1+\varepsilon)^i - (1+\varepsilon)^i)q_u$. Since $f_u$ is a sum of such approximate terms, we conclude that $f_u = (1\pm\varepsilon)\text{value}^{\uparrow}(v,(1+\varepsilon)^i)$.

Suppose we set $r_u^{\downarrow} = (1+\varepsilon)^{i-1}$ in step B.3.2.2.1. In the last iteration $i-1$, we had $f_u \le \text{f-cost}(u)$, and hence $(1-\varepsilon)\text{value}^{\uparrow}(v,(1+\varepsilon)^{i-1}) \le \text{f-cost}(u)$. However, in the current iteration $i$, we have $f_u > \text{f-cost}(u)$, and hence $(1+\varepsilon)\text{value}^{\uparrow}(u,(1+\varepsilon)^i) > \text{f-cost}(u)$. It follows that there is a value $\text{f-cost}^{\approx}(u) = (1\pm\varepsilon)\text{f-cost}(u)$ and an $r_u^{\approx} \in [(1+\varepsilon)^{i-1}, (1+\varepsilon)^i)$ such that $\text{value}^{\uparrow}(v, r_u^{\approx}) = \text{f-cost}^{\approx}(u)$. ∎

For any solution $T$, using the $r_v^{\approx}$ from Lemma 6, we define

$$\text{cost}^{\uparrow\approx}(T) = \sum_{v\in V}\text{dist}^{\uparrow}(v,T) + \sum_{f\in T}\text{f-cost}^{\approx}(f).$$

Since $\varepsilon = \Theta(1/\log n)$, we get the following.

Observation 7. *For any solution $T$, we have $\text{cost}^{\uparrow\approx}(T) = (1\pm O(\varepsilon))\text{cost}(T) = (1\pm o(1))\text{cost}(T)$.*

Algorithm C. Implements an approximate version of Phase 2, constructing the solution $S$.

```
C.1.     set $S = \emptyset$
C.2.     for $i = 0, 1, 2 \dots,$
C.2.1.       let $W$ be the set of vertices $w \in V$ with $r^{\downarrow}_w = (1+\varepsilon)^i$
C.2.2.       remove from $W$ all vertices $w$ within distance $2(1+\varepsilon)^i$ from $S$, that is,
             with $\mathrm{dist}^{\uparrow}(w, S) \le 2(1+\varepsilon)^i$
C.2.3.       using Proposition 3, construct a maximal dist-$2(1+\varepsilon)^i$ independent set
             $I$ from $W$
C.2.4.       add $I$ to $S$
C.3.     return $S$
```

The following lemma states that Algorithm C implements Phase 2 but using $r^{\downarrow}_v$ instead of $r_v$ or $r^{\approx}_v$.

LEMMA 8. *There exists a total ordering $\prec$ of the vertices in $V$ such that $u \prec v \Rightarrow r^{\downarrow}_v \ge r^{\downarrow}_u$ and such that $v$ is added to $S$ if and only if there is no previous $u \prec v$ in $S$ with* $\mathrm{dist}(u, v) \le 2r^{\downarrow}_v$.

*Proof.* The ordering is obtained if each iteration first lists the vertices in $I$ that we pick for $S$ and second lists the rest of the vertices in $W$. ∎

Using Lemmas 6 and 8, we will argue that Algorithms B and C are good approximations of Mettu and Plaxton's Phases 1 and 2. Unfortunately, we cannot just apply Lemma 5 in conjunction with Observation 7. We have two basic problems. One is that the algorithms use $r^{\downarrow}_v$ instead of $r^{\approx}_v$, and we may have $\mathrm{value}^{\uparrow}(v, r_v) \ll \text{f-cost}^{\approx}(v)$. The other problem is that $\mathrm{dist}^{\uparrow}$ does not satisfy the triangle inequality. To deal with these problems, we have to redo the analysis from [23] so as to prove the following claim.

THEOREM 9. *In near-linear expected time, w.h.p., Algorithms* B *and* C *find a factor $3 + o(1)$ approximate solution $S$ to the facility location problem.*

*Proof.* Recall that $\varepsilon = \theta(1/\log n)$. First, there are at most $\log_{1+1/\varepsilon} nN = O((\log n)(\log nN))$ possible values of $i$, and hence at most that many iterations in each algorithm. Since each iteration takes near-linear expected time, we conclude that the overall expected running time is near-linear.

The rest of the proof is a modified version of the one of Lemma 5 in [23]. The proof is divided into several local claims. In the proof, we will carefully switch between $\mathrm{dist}^{\uparrow}$ and dist, where only the latter supports triangle inequality. In particular, this concerns the proof of Claim 9.5 below.

From Lemma 8, we immediately get our first local claim.

CLAIM 9.1. *For any vertex $v \in V$, there is a vertex $u \in S$ such that $u \preceq v$ and* $\mathrm{dist}(u, v) \le 2r^{\downarrow}_v$. ∎

CLAIM 9.2. *For any distinct vertices $u, v \in S$, we have* $\mathrm{dist}(u, v) > 2\max\{r^{\downarrow}_u, r^{\downarrow}_v\}$.

*Proof.* By symmetry, we may assume that $u \prec v$. By Lemma 8, we have $\mathrm{dist}(u, v) > 2r^{\downarrow}_v \ge 2r^{\downarrow}_u$. ∎

For any solution $T$ and vertex $v \in V$, we define

$$\mathrm{charge}(v, T) = \mathrm{dist}^{\uparrow}(v, T) + \sum_{u \in T} \max\{0, r^{\approx}_u - \mathrm{dist}^{\uparrow}(u, v)\}.$$

CLAIM 9.3. *For any solution $T$,* $\sum_{v \in V} \mathrm{charge}(v, T) = \mathrm{cost}^{\uparrow\approx}(T)$.

*Proof.* We have

$$\sum_{v \in V} \mathrm{charge}(v, T) = \sum_{v \in V} \mathrm{dist}^{\uparrow}(v, T) + \sum_{u \in T} \left( \sum_{v \in V, \mathrm{dist}^{\uparrow}(u,v) \le r^{\approx}_u} (r^{\approx}_u - \mathrm{dist}^{\uparrow}(u, v)) \right)$$

$$= \sum_{v \in V} \operatorname{dist}^{\uparrow}(v, T) + \sum_{u \in T} \operatorname{value}^{\uparrow}(u, r_u^{\approx}),$$

which equals $\operatorname{cost}^{\uparrow\approx}(T)$ since $\operatorname{value}^{\uparrow}(u, r_u^{\approx}) = \text{f-cost}^{\approx}(u)$ by Lemma 6. ∎

CLAIM 9.4. *Let $v$ be a vertex and $u$ be nearest to $v$ in a solution $T$. Then* $\operatorname{charge}(v, T) \geq \max\{r_u^{\approx}, \operatorname{dist}^{\uparrow}(u, v)\}$.

*Proof.* Since $\operatorname{charge}(v, T) \geq \operatorname{dist}^{\uparrow}(v, T) = \operatorname{dist}^{\uparrow}(u, v)$, the statement is trivial if $\operatorname{dist}^{\uparrow}(u, v) \geq r_u^{\approx}$. Otherwise, $\operatorname{charge}(x, T) \geq \operatorname{dist}^{\uparrow}(u, v) + (r_u^{\approx} - \operatorname{dist}^{\uparrow}(u, v)) = r_u^{\approx} \geq \operatorname{dist}^{\uparrow}(u, v)$. ∎

CLAIM 9.5. *If $v \in V$ and $u \in S$, we have* $\operatorname{charge}(v, S) \leq \max\{r_u^{\approx}, \operatorname{dist}^{\uparrow}(v, u)\}$.

*Proof.* If there is no vertex $u' \in S$ with $\operatorname{dist}^{\uparrow}(v, u') \leq r_{u'}^{\approx}$, then $\operatorname{charge}(v, S) = \operatorname{dist}^{\uparrow}(v, S) \leq \operatorname{dist}^{\uparrow}(v, u)$. Hence, we may assume that there is a $u' \in S$ with $\operatorname{dist}^{\uparrow}(v, u') \leq r_{u'}^{\approx}$, or equivalently, with $\operatorname{dist}(v, u') \leq r_{u'}^{\downarrow}$.

Consider a vertex $w \in S$ which is different from $u'$. By Claim 9.2, we have that $\operatorname{dist}(u', w) > 2\max\{r_{u'}^{\downarrow}, r_w^{\downarrow}\}$. Hence $\operatorname{dist}(v, w) \geq \operatorname{dist}(u', w) - \operatorname{dist}(v, u') > 2\max\{r_{u'}^{\downarrow}, r_w^{\downarrow}\} - r_{u'}^{\downarrow} \geq \max\{r_{u'}^{\downarrow}, r_w^{\downarrow}\}$, but this implies that $\operatorname{dist}^{\uparrow}(v, w) > \max\{r_{u'}^{\approx}, r_w^{\approx}\}$.

We conclude that $\operatorname{charge}(v, S) = \operatorname{dist}^{\uparrow}(v, S) + r_{u'}^{\approx} - \operatorname{dist}^{\uparrow}(v, u') \leq r_{u'}^{\approx}$ and that $u = u'$ if $\operatorname{dist}^{\uparrow}(v, u) \leq r_{u'}^{\approx}$. ∎

CLAIM 9.6. *For any $v \in V$ and solution $T$,* $\operatorname{charge}(v, S) \leq 3(1 + \varepsilon)\operatorname{charge}(v, T)$.

*Proof.* Let $u \in T$ be such that $\operatorname{dist}^{\uparrow}(x, u) = \operatorname{dist}^{\uparrow}(x, T)$. By Claim 9.1, there exists a vertex $u' \in S$ with $u' \preceq u$ and $\operatorname{dist}(u, u') \leq 2r_u^{\downarrow}$.

If $\operatorname{dist}^{\uparrow}(v, u') \leq r_{u'}^{\approx}$, then $\operatorname{charge}(v, S) \leq r_{u'}^{\approx}$, by Claim 9.5. The statement now follows, since $u' \preceq u$ implies $r_{u'}^{\downarrow} \leq r_u^{\downarrow}$ and Claim 9.4 implies that $\operatorname{charge}(v, T) \geq r_u^{\approx}$.

Suppose instead that $\operatorname{dist}^{\uparrow}(v, u') > r_{u'}^{\approx}$. By Claim 9.5, we have $\operatorname{charge}(v, S) \leq \operatorname{dist}^{\uparrow}(v, u') < (1 + \varepsilon)\operatorname{dist}(v, u')$. Moreover, $\operatorname{dist}(v, u') \leq \operatorname{dist}(v, u) + \operatorname{dist}(u, u') \leq \operatorname{dist}(v, u) + 2r_u^{\downarrow}$. The result now follows from Claim 9.4 since the ratio of $\operatorname{dist}(v, u) + 2r_u^{\downarrow}$ over $\max\{r_u^{\approx}, \operatorname{dist}^{\uparrow}(x, u)\}$ is at most 3. ∎

Now Theorem 9 follows from Observation 7 in conjunction with Claims 9.3 and 9.6. ∎

Currently our solutions to the $k$-center or facility location problem are found in near-linear expected time (cf. Theorems 4 and 9). To get a solution in worst-case near-linear time, we can search $\Theta(\log^2 n)$ solutions in parallel, returning the first solution found, and give up if no search terminates within twice the expected time. Since the solutions of the original searches were "good" w.h.p., our parallel search finds a good solution w.h.p. in near-linear worst-case time.

## 5. *k*-median.

In the *k-median problem* for a metric $(P, \operatorname{dist})$, we want to find a set $S \subseteq P$ of $k$ facilities minimizing $\sum_{x \in P} \operatorname{dist}(x, S)$. The main result of this paper is that we can get a constant factor approximation in $\tilde{O}(m)$ time when the metric is the shortest path metric for a graph with $m$ edges (and $n$ vertices).

Our $\tilde{O}(m)$ solution to the $k$-median problem in graphs is rather complicated and hence unlikely to be of direct practical relevance. However, in our developments we present some theoretically weaker algorithms that are simple and easy to implement, yet provide stronger bounds than were previously known.

First, in section 5.2, we present a simple fast randomized algorithm for selecting a set of $\tilde{O}(k)$ potential facilities, guaranteed to contain a solution with $k$ facilities with at most twice the cost of the optimal solution. Using this as a preprocessing step for Jain and Vazirani's algorithm, we find a factor $12 + o(1)$ approximation in $\tilde{O}(kn)$ time for distance oracles, with efficient implementations for external memory and the streaming model, and in $\tilde{O}(km)$ time for graphs. This part is considered very practical, "perfect" for distance oracles, and good for graphs if $k$ is not too large.

Next, in section 5.4, we show that it can be meaningful to apply the algorithm from [19] to a sparse graph whose weights do not satisfy the triangle inequality, and construct a graph for which this makes sense. In $\tilde{O}(m)$ time, this process leads to a factor $12+o(1)$ approximation to the $k$-median problem but cheating using $k+k/\log^2 n$ facilities. This part is still simple enough to be of practical use, and good enough if the bound on the number of facilities is not sharp.

The true difficulty in the $k$-median problem is the sharp bound on the facilities. In section 5.5, we present a rather convoluted recursion for getting rid of the last $k/\log^2 n$ extra facilities. It is based on structural theorems showing that if we cannot easily get rid of the $k/\log^2 n$ extra facilities, it is because our current solution is very similar to any optimal solution. This similarity allows us to fix most of the facilities and recurse on a $o(k)$-median problem. As described, this last step is considered too complicated to be of practical interest, but it takes us to our theoretical goal: a near-linear time constant factor approximation to the $k$-median problem.

**5.1. Notation and terminology.** For $S \subseteq P$, define $\text{cost}(S) = \sum_{x\in P} \text{dist}(x, S)$. Then the *k-median problem* is the problem of finding $S \subseteq P$, $|S| \leq k$, minimizing $\text{cost}(S)$. Define $k\text{-mediancost}^{\subseteq F} = \min\{\text{cost}(S) : S \subseteq F, |S| = k\}$ and $k\text{-mediancost} = k\text{-mediancost}^{\subseteq P}$. By a factor $c$ approximation to the $k$-median problem, we mean a solution $S \subseteq P$, $|S| = k$, with $\text{cost}(S) \leq c \times k\text{-mediancost}$.

**5.2. Sampling $k \log^{O(1)} n$ facilities.** In this section, we will show how to sample a set $F$ of $k \log^{O(1)} n$ facilities, which we expect to contain a factor $2 + o(1)$ approximation to the $k$-median problem. The set $F$ is found by the following simple algorithm.

ALGORITHM D. Given a metric $(P, \text{dist})$ and a parameter $\varepsilon$, $0 < \varepsilon < 0.5$, with probability at least $1/2$, the algorithm constructs a set $F$ of $O(k \log^2 n/\varepsilon)$ facilities which contains a factor $2 + \varepsilon$ approximation to the $k$-median problem.

```
D.1.     $R := P$; $F := \emptyset$;
D.2.     while $R \neq \emptyset$ but for at most $3 \log n$ iterations, do
D.2.1.       add $21k(\log n)/\varepsilon$ random points from $R$ to $F$, or the rest of $R$ if $|R| \leq$
             $21k(\log n)/\varepsilon$
D.2.2.       for each point $x \in R$, compute the distance $\text{dist}(x, F)$ to the nearest
             neighbor in $F$
D.2.3.       pick a random $t \in R$, and remove from $R$ all those $x$ with $\text{dist}(x, F) \leq$
             $\text{dist}(t, F)$.
D.3.     return $F$
```

The following theorem states the significant properties of Algorithm D.

THEOREM 10. *Algorithm* D *computes a set $F$ of size $O(k(\log^2 n)/\varepsilon)$ by computing all points nearest neighbors in $O(\log n)$ subsets of $P$, each of size $O(k(\log n)/\varepsilon)$. With probability at least $1/2$, we get $k\text{-mediancost}^{\subseteq F} \leq (2+\varepsilon) \times k\text{-mediancost}$, and the $2+\varepsilon$ approximation factor holds even if the optimal solution is allowed to place facilities outside $P$.*

Note here that when comparing our solution with an optimal solution using facilities outside $P$, we may lose a factor of 2 just by restricting ourselves to $P$. For a concrete worst-case example, just take the 1-median problem where all points in $P$ are at distance 2 from each other, whereas the optimal solution is allowed to use an extra center of distance 1 to all points in $P$. It follows that our factor $2 + \varepsilon$ is close to the best possible. Allowing the optimal solution to use facilities outside $P$ is crucial to a later reduction in the proof of Lemma 13.

*Proof of Theorem* 10. The statements concerning the size of $F$ and the all points nearest neighbor computations follow directly from the description of Algorithm D as it performs at most $3\log n$ iterations of the while loop D.2. The hard part is to prove that with probability at least 1/2, $k$-mediancost$^{\subseteq F} \leq (2+\varepsilon) \times k$-mediancost.

For the analysis, we define $R_0$ and $F_0$ to be the initial values of $R$ and $F$. Also, we let $R_i$ and $F_i$ be the values after the $i$th iteration. Finally, we let $P_i$ be the points removed from $R = R_{i-1}$ in the $i$th iteration, that is, $R_i = R_{i-1} \setminus P_i$. Let $\ell$ be the final iteration. Then $F_\ell$ is the final set returned by the algorithm.

First we will argue that we expect $R = R_\ell$ to be empty when done with the while loop. In this case, all points in $P$ end in some $P_i$, so the $P_i$ partition $P$.

CLAIM 10.1. *The probability that $R_\ell \neq \emptyset$ is $o(1)$.*

*Proof.* In proving this claim, we think of Algorithm D as always performing $\ell^* = 3\log n$ iterations. However, pseudoiterations $i > \ell$ with $R_{i-1} = \emptyset$ have no effect. For a real iteration $i \leq t$, there is a probability of at least 1/2 that the iteration removes half the points in $R$. Also, all pseudoiterations reduce the size of $R$ by a factor 2 since $0/2 = 0$. Hence, for each iteration $i \leq \ell^*$, there is a probability of at least 1/2 that $|R_i| \leq |R_{i-1}|/2$. Consequently, the expected number of such iterations is at least $1.5\log n$.

On the other hand, if $R_\ell \neq \emptyset$, we have $\ell^* = \ell$ and $|R_{\ell^*}| \geq 1$. Hence there can be at most $\log|R_0| = \log n$ iterations $i$ with $|R_i| \leq |R_{i-1}|/2$, that is, only 2/3 of the expected number of such iterations. Using a standard Chernoff bound (see, e.g., [24, Theorem 4.2]), we get that the probability of this event is at most $\exp(-(1.5\log n)(1/3)^2/2) = \exp(-(\log n)/12) < n^{-0.12}$. ∎

Let $OPT$ be an optimal solution to the $k$-median problem. Our claimed good solution inside $F$ will be $OPT^F$ denoting $\{a^F\}_{a\in OPT}$. Recall there that $a^F$ is a facility $f \in F$ which is nearest to $a$. Trivially $|OPT^F| \leq |OPT| \leq k$. For our analysis, we will assign each $x \in P$ to $(x^{OPT})^F$. The cost assigned to $x$ will be

$$\mathrm{dist}(x, x^{OPT}) + \mathrm{dist}(x^{OPT}, (x^{OPT})^F).$$

We refer to $\mathrm{dist}(x^{OPT}, (x^{OPT})^F)$ as the "extra cost" of $x$. This extra cost can only decrease as $F$ grows. Our goal is to show that, with probability at least 1/2,

$$(1) \qquad \sum_{x\in P} \mathrm{dist}(x^{OPT}, (x^{OPT})^F) \leq \sum_{x\in P}((1+\varepsilon)\mathrm{dist}(x, x^{OPT})) = (1+\varepsilon)\mathrm{cost}(OPT).$$

All points in $P$ start off "unhappy." A point $x \in P$ will be defined as "happy" from the moment that a point $a$ with $\mathrm{dist}(a, x^{OPT}) \leq \mathrm{dist}(x, x^{OPT})$ is picked for $F$. From that moment, $\mathrm{dist}(x^{OPT}, (x^{OPT})^F) \leq \mathrm{dist}(x^{OPT}, a) \leq \mathrm{dist}(x, x^{OPT})$. Hence, if all points ended up happy, we would get (1) satisfied with $\varepsilon = 0$. Unfortunately, we cannot hope to make all points happy, but we will find a way to pay for the unhappy ones.

Consider an unhappy point $x$. The extra cost of $x$ is $\mathrm{dist}(x^{OPT}, (x^{OPT})^F) \leq \mathrm{dist}(x^{OPT}, x^F) \leq \mathrm{dist}(x^{OPT}, x) + \mathrm{dist}(x, x^F) \leq \mathrm{dist}(x, OPT) + \mathrm{dist}(x, F)$. Thus, to prove (1) it suffices to show that the sum over $\mathrm{dist}(x, F)$ over all unhappy $x$ is bounded by $\varepsilon\,\mathrm{cost}(OPT)$.

We say an iteration $i$ "ends well" if the following equation is satisfied at the end of iteration $i$:

$$\sum_{\text{unhappy } x\in P_i} \text{dist}(x, F_i) \;\leq\; \varepsilon \sum_{\text{happy } x\in P_i} \frac{\text{dist}(x, F_i)}{2}. \tag{2}$$

As part of our proof of Theorem 10, we will prove a series of local claims concerning individual iterations. The first states that if an iteration $i$ ends well, the extra cost of points in $P_i$ satisfies (1). Later we will argue that we expect all iterations to end well.

CLAIM 10.2. *If iteration $i$ ends well, $\sum_{x\in P_i} \text{dist}(x^{OPT}, (x^{OPT})^{F_i}) \leq \sum_{x\in P_i}((1+\varepsilon)\text{dist}(x, OPT))$.*

*Proof.* Following our discussion of happy and unhappy points, it suffices to show that

$$\sum_{\text{unhappy } x\in P_i} \text{dist}(x, F_i) \;\leq\; \varepsilon \sum_{x\in P_i} \text{dist}(x, OPT).$$

However, if $x$ is happy, we have

$$\text{dist}(x, F_i) \leq \text{dist}(x, x^{OPT}) + \text{dist}(x^{OPT}, (x^{OPT})^F) \leq 2\,\text{dist}(x, OPT).$$

Thus from (2) we get

$$\begin{aligned}\sum_{\text{unhappy } x\in P_i} \text{dist}(x, F_i) &\leq \varepsilon \sum_{\text{happy } x\in P_i} \frac{\text{dist}(x, F_i)}{2}\\ &\leq \varepsilon \sum_{\text{happy } x\in P_i} \text{dist}(x, OPT)\\ &\leq \varepsilon \sum_{x\in P_i} \text{dist}(x, OPT),\end{aligned}$$

as desired. □

The next two claims concern the probability that an iteration $i$ ends well.

CLAIM 10.3. *The expected fraction of unhappy points in $R_{i-1}$ after step* D.2.1 *is $\leq \varepsilon/(21\log n)$.*

*Proof.* Consider a point $x \in R_{i-1}$ which was not happy before step D.2.1, and let $C$ be the remaining part of the $OPT$-cluster containing $x$; that is, $C = \text{cluster}(x^{OPT}, OPT) \cap R_{i-1}$.

Suppose there are $q$ points in $C$, including $x$, that are as close to $x^{OPT}$ as $x$. Now, the probability that $x$ is not turned happy by step D.2.1 is $(1 - q/|R_i|)^{21k(\log n)/\varepsilon} < e^{-q21k(\log n)/(\varepsilon|R_{i-1}|)}$. Thus, no matter the size of $C$, the expected number of unhappy points in $C$ is at most $\sum_{q=1}^{\infty} e^{-q21k(\log n)/(\varepsilon|R|)} < \int_{x=0}^{\infty} e^{-x21k(\log n)/(\varepsilon|R_{i-1}|)}dx < \varepsilon|R_{i-1}|/(21k(\log n))$, so with $k$ clusters, the expected fraction of unhappy elements in $R_{i-1}$ is at most $\varepsilon/(21\log n)$. □

CLAIM 10.4. *Let $u$ be the fraction of unhappy points in $R_{i-1}$ after step* D.2.1. *Then the probability that iteration $i$ does not end well is at most $u(2 + 2/\varepsilon)$.*

*Proof.* Let the points in $R_{i-1}$ be sorted in order of increasing distance to $F$, resolving ties so that unhappy points come before happy points. Traverse $R_{i-1}$ and let each unhappy point grab the next $\lceil 2/\varepsilon\rceil$ happy points, if any, that are not grabbed yet. Now, if point $t$ from step D.2.3 is neither unhappy nor grabbed, then each unhappy point in $P_i$ has all its $\lceil 2/\varepsilon\rceil$ grabbed points in $P_i$, and hence (2) is satisfied.

The fraction of points that are unhappy or grabbed by an unhappy point is at most $u(\lceil 2/\varepsilon \rceil + 1)$. ∎

Claim 10.5. *The probability that iteration $i$ does not end well is at most* $1/(7 \log n)$.

*Proof.* By Claim 10.4, the probability that iteration $i$ does not end well is at most the sum over each possible fraction $u$ of unhappy points in $R_i$ of $u(2 + 2/\varepsilon)$ times the probability of the fraction $u$, but this equals $\mathsf{E}[u](2 + 2/\varepsilon)$. Moreover, by Claim 10.3, $\mathsf{E}[u] \le \varepsilon/(21 \log n)$, so we conclude that the probability of not ending well is bounded by $(\varepsilon/(21 \log n))(2 + 2/\varepsilon) \le (2 + 2\varepsilon)/(21 \log n) < 1/(7 \log n)$. The last inequality used the fact that $\varepsilon < 0.5$. ∎

We have at most $3 \log n$ iterations, and so by Claim 10.5, the probability that all iterations end well is at least

$$1 - \frac{(3 \log n)}{(7 \log n)} = \frac{4}{7}.$$

Combining this with Claim 10.1, we conclude that the probability that all iterations end well and that we finish with $R = R_\ell = \emptyset$ is at least $4/7 - o(1) > 1/2$. We now assume this combination of events.

Since we finish with $R = R_\ell = \emptyset$, the $P_i$ partition $P$. Then (1) follows directly from Claim 10.2. This completes the proof of Theorem 10. ∎

Often we prefer a high probability result, as obtained by the following algorithm.

Algorithm E. Given a metric $(P, \mathrm{dist})$, the algorithm constructs a set $F$ of $O(k \log^4 n)$ facilities which, w.h.p., contains a factor $2 + o(1)$ approximation to the $k$-median problem.

```
E.1.    for $i = 1, \ldots, \lceil \log^{3/2} n \rceil$,
E.1.1.      with $\varepsilon = 1/\sqrt{\log n}$, apply Algorithm D to $(P, \mathrm{dist})$, obtaining a set $F_i$
E.2.    return $F = \bigcup_i F_i$
```

Theorem 11. *Algorithm* E *involves $O(\log^{2.5} n)$ computations of nearest neighbors in sets of size $O(k \log^{1.5} n)$, and the final set $F$ has size $O(k \log^4 n)$. Moreover, w.h.p., $k$-mediancost$^{\subseteq F} \le (2 + o(1)) \times k$-mediancost. The $2 + o(1)$ approximation factor holds even if the optimal solution is allowed to place facilities outside $P$.*

*Proof.* By Theorem 10, with probability at least $1 - 1/2^{\log^{3/2} n} = 1 - 1/n^{\omega(1)}$, there is some set $F_i$ with $k$-mediancost$^{\subseteq F_i} \le (2 + 1/\sqrt{\log n}) \times k$-mediancost. Now the result follows because $k$-mediancost$^{\subseteq F} \le k$-mediancost$^{\subseteq F_i}$ for all $i$. ∎

## 5.3. Distance oracles, external memory, and the streaming model.

The simple sampling above has several interesting consequences for distance oracles, with or without memory restrictions.

Proposition 12. *For a metric $(P, \mathrm{dist})$, w.h.p. we can find a factor $(12 + o(1))$ approximation to the $k$-median problem, using $\tilde{O}(kn)$ distance queries and computation time, and $\tilde{O}(n)$ space. The $12 + o(1)$ approximation factor holds even if the optimal solution is allowed to place facilities outside $P$.*

*Proof.* By Theorem 11, we spend $\tilde{O}(kn)$ distance queries on finding a set of $\tilde{O}(k)$ relevant facility locations, and then the algorithm from [19] allows us to solve the $k$-median problem in $\tilde{O}(kn)$ time and space. To get the $\tilde{O}(n)$ space, we actually need the full power of the techniques presented in this paper, as stated in Theorem 32. ∎

The previous $\tilde{O}(kn)$ algorithm by Guha et al. [12] had an approximation factor of $300 + o(1)$ for unbounded $k$ and $80 + o(1)$ if $k = O(\sqrt{n})$.

We will now demonstrate that our techniques give interesting results for various kinds of restricted memory, including the streaming model which was the target of [12]. We will use the following generic algorithm.

ALGORITHM F. Generic $k$-median algorithm for metric $(P, \mathrm{dist})$, which provides a factor $38 + o(1)$ approximation w.h.p.

```
F.1.    Compute the set $F$ using Algorithm E.
F.2.    Construct the multiset $P^F = \{x^F\}_{x \in P}$.
F.3.    Apply Proposition 12 to $P^F$ and return the found solution $S$.
```

Above $P^F$ is the set $F$, but where each $f \in F$ has a multiplicity or weight equal to the number of points $x \in P$ with $x^F = f$. We do not remember which points in $P$ gave rise to the multiplicities, so $P^F$ is stored in $O(|F|) = \tilde{O}(k)$ space. The set $F$ will be computed using some variant of Algorithm D. We place $F$ in internal memory, and then we compute $P^F$ by a linear scan of $P$, taking each point $x \in P$, an identifying $x^F$ in $O(|F|) = \tilde{O}(k)$ time. Hence, given $F$, the construction of $P^F$ takes $\tilde{O}(kn)$ time and $\tilde{O}(k)$ space. Finally Proposition 12 computes $S$ for $P^F$ internally in $\tilde{O}(k^2)$ time and $\tilde{O}(k)$ space.

LEMMA 13. *Algorithm* F *is a factor* $38 + o(1)$ *approximation to the k-median problem for P.*

*Proof.* Let $OPT$ be the optimal solution to the original $k$-median problem over $P$. Now, considering $OPT$ as a solution for $P^F$, the cost is $\mathrm{cost}_{P^F}(OPT) \le \sum_{x\in P} \mathrm{dist}(x^F, x^{OPT}) \le \sum_{x\in P} \mathrm{dist}(x^F, x) + \sum_{x \in P} \mathrm{dist}(x, x^{OPT}) \le k\text{-mediancost}_P^{\subseteq F} + \mathrm{cost}_P(OPT) \le (3 + o(1))\mathrm{cost}_P(OPT)$. We now apply Proposition 12 to $P^F$, getting a solution $S$ with $\mathrm{cost}_{P^F}(S) \le (12 + o(1))\mathrm{cost}_{P^F}(OPT) \le (36 + o(1))\mathrm{cost}(OPT)$. Finally, $\mathrm{cost}_P(S) \le \sum_{x\in P} \mathrm{dist}(x, x^F) + \sum_{x \in P} \mathrm{dist}(x^F, (x^F)^S) = \mathrm{cost}_P(F) + \mathrm{cost}_{P^F}(S) \le (38 + o(1))\mathrm{cost}(OPT)$.

Note even if $OPT \subseteq P$, we may have $OPT \not\subseteq F$, and hence it is important that Proposition 12 does not require that $OPT$ be a subset of the point set. □

Our first restricted memory model is external memory which is divided into blocks. Instead of counting individual accesses to external memory cells, we count the number of read and writes of blocks.

PROPOSITION 14. *W.h.p., in the external memory model with block-size B and access to a distance oracle, if n points are stored in n/B blocks, we can find a factor* $38 + o(1)$ *approximation to the k-median problem using* $\tilde{O}(kn)$ *time and distance queries,* $\tilde{O}(k)$ *internal space, and* $\tilde{O}(n/B)$ *blocks and block operations.*

*Proof.* We simply note that Algorithm D fits perfectly with external memory. In each iteration, we first scan the current set $R$, picking out the new points for $S$ and the point $t$. Both $S$ and $t$ are stored in internal memory. We now make another scan creating the new set $R'$ with the points from $R$ further from $S$ than $t$. Here both $R$ and $R'$ are packed in blocks, so the iteration takes $O(|R|/B)$ block operations. In expectation $R$ decreases by a constant factor in each iteration, and w.h.p., the total number of blocks and block operations is $O(n/B)$. □

Next, we consider a model where we can modify only our internal space, but where we have arbitrary access to the input. This is the model in which log-space is defined as having logarithmic internal space.

PROPOSITION 15. *In Proposition* 12*, the internal space can be reduced to* $\tilde{O}(k)$ *at the expense of increasing the approximation factor to* $38 + o(1)$.

*Proof.* First, we assume that we have access to a random permutation of the points in $P$. We can then implement Algorithm D in a single scan of $P$. From each iteration $j$, we will store the set $T_j$ of new points picked for $S$, the threshold point $t_j$, and the distance $\delta_j$ from $t_j$ to $S_j = \bigcup_{i \le j} T_i$. Now, iteration $j$ is implemented as follows. We continue our scanning of the points from $P$ from where we stopped in the last iteration. A point $x$ is *distant* if for each $i < j$, $\mathrm{dist}(x, S_i) \le \delta_i$. We now build $T_j$ from the next $k/\varepsilon \log^2 n$ distant points, and the following distant point is our new threshold point $t_j$. Finally, we compute $\delta_j = \operatorname{dist}(t_j, S_j)$. The distant points are exactly the points removed from $R$ by Algorithm D, so this is just a new implementation.

To check whether a point $x$ is distant, we do as follows. First we set $d := 0$. Then, for : $i = 1, \ldots, j-1$, we set $d := \min\{d, \operatorname{dist}(x, T_i)\}$, and if $d \leq \delta_i$, we conclude that $x$ is not distant.

In case we do not have access to a random permutation, then we pick $O(n \log n)$ points randomly from $P$, thus making sure that all points are picked eventually. We note that a point will not be distant the second time it is picked, so we do not need to worry about the repetitions. □

In [12], the focus was on the streaming model, where points are stored in a read-only stream that can only be read sequentially from one end to the other, having only limited internal memory. The stream could, for example, be a tape. The main performance measure is the number of times we read over the tape. The main result of [12] is that with $\tilde{O}(kn^{1/\ell})$ internal memory, we can find a $2^{O(\ell)}$ approximation for the $k$-median problem reading the stream once using $\tilde{O}(kn)$ distance queries and internal computation time. Our sampling can directly improve and simplify the techniques from [12]. Their essential step is that they take a set $X \subseteq P$, $|X| \gg k$, points and find a set $F$ of roughly $k$ facilities such that if $OPT$ is the optimal $k$-median solution, the cost of assigning the points in $X$ to $F$ is a constant factor from the cost of assigning the points in $X$ to $OPT$. For this step they use a combination of techniques from [1, 15, 19], and the exact approximation factor is unclear but appears to be large. However, the sampling technique of Theorem 11 does this job directly, with a near-optimal approximation factor of $2 + o(1)$.

More interestingly, we can replace the exponential factor $2^{O(\ell)}$ with a constant, and instead just pay a linear factor $O(\ell)$ in time.

PROPOSITION 16. *W.h.p., in the distance oracle streaming model, with $O(kn^{1/\ell})$ internal memory, we can find a $38 + o(1)$ approximation for the $k$-median problem reading the stream $O(\ell)$ times using $\tilde{O}(kn)$ distance queries and internal computation time.*

*Proof.* We are going to emulate the proof of Proposition 15. Each "streaming round" will correspond to a number of iterations of Algorithm D. In each streaming round we first pick a random set $X$ of $O(kn^{1/\ell})$ points from $P$ that are still distant. To do this, we first make one scan to count the number of distant points. We then select $O(kn^{1/\ell})$ random indices of distant points, and finally, we scan $P$ again, picking out the indexed points. We now continue with the iterations of Algorithm D, as described in the proof of Proposition 15 but taking the random points from $X$. Since we know that we end up picking only $\tilde{O}(k)$ facilities for $S$, in order to run out of points in $X$, we must be skipping all but a fraction $\tilde{O}(k)/O(kn^{1/\ell}) = \tilde{O}(1/n^{\ell})$ of the points. Since $X$ is randomly chosen from points distant at the start of the streaming round, we conclude w.h.p. that the fraction of distant points has dropped by a factor $\tilde{\Omega}(n^{\ell})$. Hence there are only $O(\ell)$ streaming rounds. □

## 5.4. Reducing to $k + k/\log^2 n$ facilities.

In this section, we will show that we can get down from the $k \log^{O(1)} n$ potential facilities from Theorem 11 to a solution $S$ with $k + k/\log^2 n$ facilities. We wish to apply the techniques of Jain and Vazirani [19], but these techniques are quoted only as working with graphs satisfying the triangle inequality. More precisely, let $F$ be the set of potential facilities. They assume all edges in $F \times P$, and that each edge $(a, x) \in F \times P$ has length $\ell(a, x) = \operatorname{dist}(a, x)$. If

$|F| = \log^{\omega(1)} n$, this is too much for our time bounds.

We will now consider what happens when the algorithm from [19] is applied when $G$ does not satisfy the triangle inequality. Define $\ell\text{-cost}(S) = \sum_{x \in P} \ell(x, x^S)$. Here $\ell(x, x^S) = \infty$ if $(x, x^S)$ is not an edge in $G$. What the algorithm from [19] really finds is a $k$-median $S \subseteq F$ with $\text{cost}(S) \le 6 \times k\text{-median-}\ell\text{-cost}^{\subseteq F}$. The point is that the dual variables providing the lower bound in [19] do not require the triangle inequality. The triangle inequality is used only in bounding the gap between the primal solution and the dual variables.

Concerning speed, the algorithm in [19] works in $\tilde{O}(m)$ time. Here we use the randomized rounding from [19, section 3.2]. For that rounding, we have two sets $A$ and $B$ of facilities, and for each $a \in A$, we need to find a nearest facility $a^B \in B$, but by Observation 1, this is done in $\tilde{O}(m)$ time. Because of the randomization, the cost is only expected.

THEOREM 17. *In $\tilde{O}(m)$ time, we can find $S \subseteq F$, $|S| = k$, with an expected cost of* $\text{cost}(S) \le 6 \times k\text{-median-}\ell\text{-cost}^{\subseteq F}$.

To apply this theorem, we construct a graph $G^\ell$ with $\tilde{O}(n)$ edges $(v, w)$, each with $\ell(v, w) = \text{dist}(v, w)$, and with $k^\ell\text{-median-}\ell\text{-cost}_{G^\ell}^{\subseteq F} = O(k\text{-mediancost}^{\subseteq F})$, where $k^\ell = k + k \log^2 n$. Applying Theorem 17 then gives us a $k^\ell$-median $S \subseteq F$ with $\text{cost}(S) \le \text{cost}_{G^\ell}(S) = O(k\text{-mediancost}^{\subseteq F})$.

The construction of $G^\ell$ based on $F$ is rather simple. Set $d = \log^3 n |F|/k = \log^{O(1)} n$. For each point $x \in P$, we include an edge to each of the $d$ nearest neighbors in $F$.

LEMMA 18. *The graph $G^\ell$ has $(k + k/\log^2 n)$-median $S \subseteq F$ with* $\ell\text{-cost}(S) \le k\text{-mediancost}^{\subseteq F}$.

*Proof.* All we need is to show the existence of a set $D \subseteq F$ of size $k/\log^2 n$ which dominates in the sense that each $x$, $x^D$ is one of its $d$ nearest neighbors. We then set $S = OPT \cup D$, where $OPT$ is the optimal solution. If $x^{OPT}$ is one of the $d$ nearest neighbors in $F$, $x^S = x^{OPT}$. Otherwise, $x^S = x^D$ and then $\ell(x, x^D) \le \text{dist}(x, OPT)$, so $\ell\text{-cost}(S) \le \text{cost}(OPT)$.

To show the existence of $D$, we just pick $D$ randomly. For each $x \in P$, the probability that none of its $d$ nearest neighbors are picked is $\le (1 - |D|/|F|)^d < e^{-\log n} < 1/n$, so there is a positive probability that this does not happen for any $x$. ◻

For the construction of $G$, we need the following result.

LEMMA 19. *W.h.p., using $O(d \log n)$ all points nearest marked neighbor computations, we can find the $d$ nearest neighbors in $F$ to each point $x$.*

*Proof.* We pick each $a \in F$ with probability $1/(2d)$ for a set $Q$ of marked neighbors. For each $x \in P$ and each $i \le d$, the probability that $x^Q$ is the $i$th nearest neighbor in $F$ is $\ge (1 - 1/(2d))^{i-1}/2d \ge 1/(4d)$. Hence, in $O(d \log n)$ all points nearest marked neighbor computations, we can find the $d$ nearest neighbors of all $x$ w.h.p. ◻

THEOREM 20. *With probability $1 - O(1/n)$, in $\tilde{O}(m)$ time, we can construct a $k + k/\log^2 n$-median $S$ with* $\text{cost}(S) \le (12 + o(1)) \times k\text{-mediancost}$.

*Proof.* First using Theorem 11, with probability at least $1/2$, we identify a set $F$ of size $k \log^{O(1)} n$ with $k\text{-mediancost}^{\subseteq F} \le (2 + o(1))k\text{-mediancost}$. Then, using the above lemmas, we construct $G^\ell$ with $(k + k/\log^2 n)\text{-median-}\ell\text{-cost}_{G^\ell} \le k\text{-mediancost}^{\subseteq F}$ and finally, we apply Theorem 17 to $G^\ell$ to get a set $S$, $|S| = k + k/\log^2 n$, with an expected cost of $(12 + o(1)) \times k\text{-mediancost}$. ◻

**5.5. Recursing down to $k$ facilities.** Let $OPT$ denote some optimal solution to the $k$-median problem. Our starting point is a solution $S$, as from Theorem 20, of satisfactory cost, that is, $\mathrm{cost}(S) = O(OPT)$, but which uses $q = k/\log^2 n$ too many facilities. We will then first try greedily to remove $q$ facilities from $S$. If this cannot be done at cost $o(\mathrm{cost}(S))$, we will be able to fix many universally good facilities and then recurse.

**5.6. Some basic definitions.** We will use $\alpha = \omega(1) \cup \log^{o(1)}(n)$ to denote a value that grows slowly with $n$, e.g., like an inverse of Ackermann's function.

For our presentation, it is useful for each $a \in S$ to define $\mathrm{clustercost}(a, S) = \sum_{x\in \mathrm{cluster}(a,S)} \mathrm{dist}(x, a)$. Thus, $\mathrm{cost}(S) = \sum_{a\in S} \mathrm{clustercost}(a, S)$. Also, we define $\delta_{a,S} = \mathrm{dist}(a, S \setminus \{a\})$ and $\mathrm{shiftcost}(a, S) = |\mathrm{cluster}(a, S)| \times \delta_{a,S}$. Note that if $b$ is nearest to $a$ in $S$, $\mathrm{shiftcost}(a, S)$ is a trivial upper-bound on the increase in cost encountered by reassigning all $x \in \mathrm{cluster}(x, S)$ to $b$; that is, for $x \in \mathrm{cluster}(a, S)$, $\mathrm{dist}(x, b) - \mathrm{dist}(x, a) \leq \mathrm{dist}(a, b) = \delta_{a,S}$.

Let $a_1, \ldots , a_{|S|}$ be an enumeration of the facilities in $S$ in order of nondecreasing $\mathrm{shiftcost}(\cdot, S)$.

**5.7. Greedy elimination of facilities.** We will now present the following routine.

Greedy$(q, S)$ eliminates $q$ terminals from $S$, increasing the cost by $\mathrm{greedycost}(q, S) \leq \sum_{i\leq 2q} \mathrm{shiftcost}(a_i, S)$.

We consider Greedy successful if $\mathrm{greedycost}(q, S) \leq \mathrm{cost}(S)/\alpha$. Then $\mathrm{cost}(\mathrm{Greedy}(q, S)) = (1 + o(1))\mathrm{cost}(S)$, and we return Greedy$(q, S)$.

To implement Greedy, we will use the following claim.

Lemma 21. *Suppose that each element $a$ in $A$ has at most one successor $s(a) \in A$. Then, in $O(|A|)$ time, we can mark at least $|A|/2$ elements so that if $a$ is marked, $s(a)$ is unmarked.*

*Proof.* Let $P(a) = \{b \in A : s(b) = a\}$. First, we mark all $a$ with $P(a) = \emptyset$. Next, repeatedly, we mark facilities $a \in A$ such that $|P(a)| = 1$, protecting $|P(a)|$ from being marked in a later round. To see that we do not run out of elements, note that on the average, $|P(a)| \leq 1$. Thus, if there are $i$ elements with $P(a) = \emptyset$, there are at most $i$ elements with $|P(a)| > 1$. Each marking of an element $a$ with $|P(a)| = 1$ protects one element from being marked, so we will mark at least half of these elements, leading to a total marking of at least $i + (|A| - 2i)/2 = |A|/2$ elements. □

To implement Greedy, we apply Lemma 21 with $A = \{a_1, \ldots , a_{2q}\}$ and $s(a)$ denoting the facility in $A$ nearest to $a$. We can then eliminate any $q$ marked facilities.

**5.8. When Greedy does not work.** We will now start studying properties of our solution when Greedy does not report success, that is, when $\mathrm{greedycost}(q, S) > \mathrm{cost}(S)/\alpha$. We say that $a \in S$ is *concentrated* if $\mathrm{clustercost}(a, S) \leq \mathrm{shiftcost}(a, S)/\alpha$.

Lemma 22. *If there are $2\alpha^2 q$ facilities in $S$ that are not concentrated, Greedy$(q, S)$ reports success.*

*Proof.* Let $A$ be the set of facilities in $S$ that are not concentrated. Then $\mathrm{greedycost}(q, S) \leq \sum_{i\leq 2q} \mathrm{shiftcost}(a_i, S) \leq \frac{2q}{|A|} \sum_{a\in A} \mathrm{shiftcost}(a, S) < \alpha^{-2} \sum_{a\in A}(\alpha \mathrm{clustercost}(a, S)) \leq \mathrm{cost}(S)/\alpha$. □

Note that if $y \in P \setminus \mathrm{cluster}(a, S)$, $\mathrm{dist}(y, a) \geq 0.5\, \delta_{a,S}$. Define $\Delta(a, S) = \{x \in P : \mathrm{dist}(x, a) \leq (0.25 - 1/\alpha)\delta_{a,S}\}$. It is easy to see that if $a$ is concentrated, most facilities in $\mathrm{cluster}(a, S)$ are inside $\Delta(a, S)$.

We say that a solution $T$ *hits* $a \in S$ if $T \cap \Delta(a, S) \neq \emptyset$; otherwise $T$ *misses* $a$.

Observation 23. *If $T$ misses $a \in S$, then* $\sum_{x\in \mathrm{cluster}(a,S)} \mathrm{dist}(x, T) \geq \sum_{x\in \mathrm{cluster}(a,S)}(\mathrm{dist}(a, T) - \mathrm{dist}(x, a)) \geq (0.25 - 1/\alpha)\, \mathrm{shiftcost}(a, S) - \mathrm{clustercost}(a, S)$, *which is* $\geq (0.25 - 2/\alpha)\mathrm{shiftcost}(a, S)$ *if $a$ is concentrated.*

As a consequence of Observation 23, if $a$ is concentrated but missed by $T$ and we add $a$ to $T$, we reduce the cost by $(0.25 - 3/\alpha)\,\mathrm{shiftcost}(a, S)$.

LEMMA 24. *If $T$ is a solution with $\mathrm{cost}(T) = O(\mathrm{cost}(OPT))$ and $T$ misses $3\alpha^2 q$ facilities in $S$, Greedy$(q, S)$ reports success.*

*Proof.* Among the $3\alpha^2 q$ misses, we may assume that there are $\geq \alpha^2 q$ concentrated facilities, for otherwise Greedy$(S)$ reports success by Lemma 22.

Let $M \subseteq S$ be the set of $\geq \alpha^2 q$ concentrated facilities missed by $T$. By Observation 23, $\mathrm{cost}(T) \geq \sum_{a\in M} \sum_{x\in \mathrm{cluster}(a,S)} \mathrm{dist}(x, T) = \sum_{a\in M} (0.25-2/\alpha)\,\mathrm{shiftcost}(a, S)$ $\geq \frac{(0.25-2/\alpha)|M|}{2q} \sum_{i\leq 2q} \mathrm{shiftcost}(a_i, S) \geq \Omega(\alpha^2)\mathrm{greedycost}(q, S)$. Now, $\mathrm{cost}(OPT) \leq \mathrm{cost}(S) + \mathrm{greedycost}(q, S) = \mathrm{cost}(S) + o(\mathrm{cost}(T)/\alpha) = \mathrm{cost}(S) + o(OPT/\alpha)$, and thus $\mathrm{greedycost}(q, S) = o(\mathrm{cost}(S)/\alpha)$. ∎

**5.9. The basic recursion.** As hinted at by Lemma 24, if Greedy$(q, S)$ is not successful, no good solution can be too far away from our current solution $S$. We will exploit this recursively by fixing most of the facilities with high shiftcost.

In order to code the fixing of facilities, we introduce the notion of a *free facility*, which is a facility which is automatically included in any $k$-median solution, but which does not count as one of the $k$ medians. In addition, we want to encode a penalty for not using a fixed facility anyway. This is done by introducing a *single* point $s$ whose only link is to a free facility $f$. Thus the assignment cost of $s$ is $\ell(s, f)$ if $s$ is not picked as a facility, and 0 otherwise.

We are now ready to describe the basic mechanics of our general recursion, but ignoring that we may have free facilities and singles in our own input.

ALGORITHM G. Solves $k$-median problem assuming no free facilities or singles in input.

```
G.1.    Construct $S \subseteq P$, $|S| \leq k + q$, $q = k/\log^2 n$ with $\mathrm{cost}(S) = O(k\text{-mediancost})$
        (cf. sections 5.2 and 5.4).
G.2.    Let $a_1, \ldots, a_{|S|}$ be the facilities in $S$ in order of increasing shiftcost (cf. sec-
        tion 5.7).
G.3.    If $\mathrm{greedycost}(q, S) \leq \mathrm{cost}(S)/\alpha$ (cf. section 5.7), return Greedy$(q, S)$ and exit
        the recursion.
G.4.    Produce a modified graph $G'$, fixing more than half the facilities as follows.
G.4.1.      Set $U = S \setminus (NOTCONC \cup SMALLSHIFT \cup LARGE)$, where
            $NOTCONC$ is the set of nonconcentrated facilities, $SMALLSHIFT =$
            $\{a_1, \ldots, a_r\}$ with $r = \alpha^4 q \log n$, and $LARGE = \{a \in S : \mathrm{clustercost}(a, S)$
            $\geq \frac{\mathrm{cost}(S)}{(\alpha^3 q \log n)}\}$.
            Define smallshiftcost = $\mathrm{shiftcost}(a_r, S)$.
G.4.2.      Using the technique from [18] (cf. the appendix), for each $a \in U$ find a
            factor $(1 + 1/\alpha)$ approximate 1-median $a'$ for $\Delta(a, S)$. If $a$ is a better
            1-median for $\Delta(a, S)$ than $a'$, set $a' := a$. We are going to use $a'$ in our
            final solution.
            Note that for distances over $\Delta(a, S)$ we need to consider only paths
            inside cluster$(a, S)$. Hence our approximate 1-medians are found based
            on disjoint induced subgraphs, so the total running time is $\tilde{O}(m)$.
G.4.3.      Create a free facility $f$ with a zero length edge to each $a'$, $a \in U$. Reduce
            all distances from $f$ by a factor 2; that is, first compute all shortest paths
            from $f$, and second add an edge to each vertex of length half the found
            distance.
G.4.4.      Introduce a set $AUX$ of $p_{\max} = 3\alpha^2 q$ singles, each connected to $f$ by an
```

```
            edge of length $(0.25 - 2.5/\alpha)$smallshiftcost.
G.4.5.      Set $k' := k - |U| + p_{\max}$. Since $q = k/\log^2 n$, $k' = q\alpha^{O(1)} \log n = o(k)$.
        In section 5.10 it will be shown that the modified graph $G'$ admits a $k'$-median
        at cost $(1 + o(1/\log n))$ times the minimal cost of a $k$-median for $G$.
G.5.    Recursively, find a $k'$-median $S'$ for the modified graph.
G.6.    Using the knowledge of $S$, we modify $S'$ into a $k$-median solution $S^*$ for the
        original graph $G$, as follows.
G.6.1.      Set $S'' = (S' \cup \{a'\}_{a\in U}) \setminus (\{f\} \cup AUX)$. All points that were assigned
            to $f$ are now assigned to their nearest facility $a'$.
            Then $S'' \subseteq P$ and $|S''| = k + p$, where $p = |AUX \setminus S'|$.
G.6.2.      Reduce $S''$ to $S^*$ by removal of $p$ facilities from $\bigcup_{a\in SMALLSHIFT} \Delta(a, S)$
            as described in section 5.11, at cost $p(2.5 - 2/\alpha)$ smallshiftcost+
            $o(\text{cost}(S)/\log n)$.
G.7.    Return $S^*$.
```

To understand the basics of the above algorithm, first note that we lose a factor $(1+o(1/\log n))$ both in step G.4 and in step G.6.2. However, the recursion has depth $o(\log k)$, so the total loss due to these factors is $(1 + o(1/\log n))^{o(\log k)} = (1 + o(1))$.

Now, since we end up using all the facilities $a'$, the reader may wonder why we even bother introducing the singles in $AUX$ that we eliminate anyway. The direct reason is that if we did not have any singles in $AUX$ and just sought a $k-|U|$ median recursively, then an optimal solution for the modified solution could be a constant factor worse than that of the original optimal solution, and hence we would lose a constant factor in each recursive step. Another obstacle is that for step G.6.2 we actually use the knowledge of $S$ to deal with the singles, and there is no obvious way of including this knowledge recursively.

As the recursion stands, our main loss is when we return from the recursion and modify $S'$ into $S^*$. In step G.6.1, each $x$ previously assigned to $f$ gets assigned to a facility $a'$ that is twice as far away. However, for each point $x$, such a reassignment will happen only once throughout the recursion. Similarly, for the singles, we lose a factor $\phi = (2.5 - 2/\alpha)/(0.25 - 2.5/\alpha) = 10 + o(1)$ in cost when they get eliminated, and again this happens only once for each single.

## 5.10. Cheap fixing of facilities.

In this section, we will show the following.

THEOREM 25. *For step* G.4*, there is a $k'$-median $OPT'$ for $G'$ with* $\text{cost}_{G'}(OPT') \leq (1 + o(1/(\alpha \log n)))\text{cost}(OPT)$*, where $OPT$ is an optimal $k$-median for $G$.*

*Proof.* Let $p$ be the number of facilities in $S$ not hit by $OPT$. By Lemma 24 with $T = OPT$, $OPT$ misses at most $3\alpha^2 q = p_{\max}$ facilities. We then obtain $OPT'$ from $OPT$ by first deleting all facilities in $\bigcup_{a\in U} \Delta(a, S)$, thus deleting at least $|U|-p$ facilities from $OPT'$. Second we add $|AUX| - p$ facilities from $AUX$. As a result, $OPT'$ gets at most $k - (|U| - p) + |AUX| - p = k'$ nonfree facilities, as desired.

We will now reassign points from $OPT$ to $OPT'$, attributing the reassignments, and their costs, to the different $a \in U$:

- If $a$ is not hit by $OPT$, we reassign each $x \in \Delta(a, S)$ to $f$. Further, we assign one point $a^s$ in $AUX \setminus OPT'$ to $f$, at cost $(0.25 - 2.5/\alpha)$smallshiftcost.
- If $a$ is hit by $OPT$, all points $x$ with $x^{OPT} \in \Delta(a, S)$ or $x \in \Delta(a, S)$ are reassigned to $f$.

Before analyzing the cost of the reassignments, we need some simple observations. (This is all part of our proof of Theorem 25.) First, as a general strengthening of Observation 23, we have the next claim.

CLAIM 25.1. *If* $\text{dist}(a, T) \geq \beta\, \delta_{a,S}$ *and* $\beta \leq 0.25 - 1/\alpha$, $\sum_{x\in\Delta(a,S)} \text{dist}(x, T) \geq$

$\beta\,\text{shiftcost}(a,S) - \text{clustercost}(a,S)$.

*Proof.* We see $\sum_{x\in\Delta(a,S)} \text{dist}(x,T) \ge \sum_{x\in\Delta(a,S)}(\text{dist}(a,T) - \text{dist}(a,x)) \ge \sum_{x\in\Delta(a,S)}(\beta\,\delta_{a,S} - \text{dist}(a,x)) \ge \sum_{x\in\text{cluster}(a,S)}(\beta\,\delta_{a,S} - \text{dist}(a,x)) \ge \beta\,\text{shiftcost}(a,S) - \text{clustercost}(a,S)$. In the third inequality above we used that $\beta\,\delta_{a,S} - \text{dist}(a,x)$ is negative for $x \notin \Delta(a,S)$. ∎

CLAIM 25.2. $\text{dist}(a,a') \le 2\delta_{a,S}/\alpha$.

*Proof.* If $\text{dist}(a,a') > 2\delta_{a,S}/\alpha$, by Claim 25.1, with $\beta = \frac{2}{\alpha}$, $\sum_{x\in\Delta(a,S)} \text{dist}(x,a') > 2\,\text{shiftcost}(a,S)/\alpha - \text{clustercost}(a,S) \ge \text{clustercost}(a,S) \ge \sum_{x\in\Delta(a,S)} \text{dist}(x,a)$, contradicting that $a'$ is at least as good a median for $\Delta(a,S)$ as $a$. ∎

CLAIM 25.3. $\sum_{x\in\text{cluster}(a,S)} \text{dist}_{G'}(x,f) \le \text{clustercost}(a,S)/(2-o(1))$.

*Proof.* First note that $\sum_{x\in\Delta(a,S)} \text{dist}_{G'}(x,f) = \sum_{x\in\Delta(a,S)} \text{dist}(x,a')/2 \le \sum_{x\in\Delta(a,S)} \text{dist}(x,a)/2$. However, if $x \in \text{cluster}(a,S) \setminus \Delta(a,S)$, using Claim 25.2, $\text{dist}(x,a') \le \text{dist}(x,a) + \text{dist}(a,a') \le \text{dist}(x,a) + 2\delta_{a,S}/\alpha \le (1+o(1))\text{dist}(x,a)$. Hence $\text{dist}_{G'}(x,f) \le \text{dist}(x,a')/2 \le \text{dist}(x,a)/(2-o(1))$. ∎

We are now ready to account for the change in cost due to the reassignments in connection with each $a \in U$.

*Suppose that OPT did not hit a.* First we have an added increase of $(0.25 - 2.5/\alpha)$ smallshiftcost for assigning $a^s$ to $f$. The only other points that get reassigned are points in $\Delta(a,S)$. By Claim 25.1 with $\beta = 0.25 - 1/\alpha$, $OPT$ paid $\sum_{x\in\Delta(a,S)} \text{dist}(x,OPT) \ge (0.25-1/\alpha)\,\text{shiftcost}(a,S) - \text{clustercost}(a,S)$. However, we only pay $\sum_{x\in\Delta(a,S)} \text{dist}_{G'}(x,f) \le \sum_{x\in\Delta(a,S)} \text{dist}(x,a)/2 \le \text{clustercost}(a,S)/2$, so the reassignments to $f$ buys us at least $(0.25-1/\alpha)\,\text{shiftcost}(a,S) - 1.5\text{clustercost}(a,S) \ge (0.25-2.5/\alpha)\,\text{shiftcost}(a,S) \ge (0.25-2.5/\alpha)$ smallshiftcost, matching the new assignment cost for $a^s$.

*Suppose that OPT did hit a.* First, we argue that if $x \notin \text{cluster}(a,S)$, the assignment cost for $x$ can only go down. We are reassigning $x \notin \text{cluster}(a,S)$ only if $x^{OPT} \in \Delta(a,S)$.

CLAIM 25.4. *If* $x \notin \text{cluster}(a,S)$ *and* $x^{OPT} \in \Delta(a,S)$, $\text{dist}_{G'}(x,f) \le \text{dist}(x,OPT)$.

*Proof.* Define $\gamma \ge 0$ by $\text{dist}(x,a) = (0.5+\gamma)\delta_{a,S}$. Then $\text{dist}(x,OPT) \ge \text{dist}(x,a) - \text{dist}(a,x^{OPT}) \ge (0.25+\gamma+1/\alpha)\delta_{a,S}$, and using Claim 25.2, $\text{dist}(x,a') \le \text{dist}(x,a) + \text{dist}(a,a') \le (0.5+\gamma+2/\alpha)\delta_{a,S}$. Hence $\text{dist}(x,f) = \text{dist}(x,a')/2 \le \text{dist}(x,a)$. ∎

Thus we do well on all $x \notin \text{cluster}(a,S)$. To complete the proof, we further divide in cases depending on whether or not $OPT$ has exactly one facility in $\text{cluster}(a,S)$.

*Suppose that OPT has* one *facility in* $\text{cluster}(a,S)$. Since $OPT$ hits $a$, this facility is $a^{OPT} \in \Delta(a,S)$. Having dealt with $x \notin \text{cluster}(a,S)$, it suffices to demonstrate an improvement over $x \in \text{cluster}(a,S)$. First we show we are done if $a^{OPT}$ is not close to $a$.

CLAIM 25.5. *If* $\text{dist}(a,a^{OPT}) > 2\delta_{a,S}/\alpha$, *then* $\sum_{x\in\text{cluster}(a,S)} \text{dist}_{G'}(x,f) < \sum_{x\in\text{cluster}(a,S)} \text{dist}(x,OPT)$.

*Proof.* By Claim 25.1 with $\beta = 2/\alpha$, we have $\sum_{x\in\text{cluster}(a,S)} \text{dist}(x,OPT) \ge \sum_{x\in\Delta(a,S)} \text{dist}(x,OPT) > 2\,\text{shiftcost}(a,S)/\alpha - \text{clustercost}(a,S)$. However, by Claim 25.3, our cost is $\sum_{x\in\text{cluster}(a,S)} \text{dist}_{G'}(x,OPT) < \text{clustercost}(a,S)$, and since $a$ is concentrated, $\text{shiftcost}(a,S)/\alpha \ge \text{clustercost}(a,S)$. ∎

CLAIM 25.6. *If* $\text{dist}(a,a^{OPT}) \le 2\delta_{a,S}/\alpha$, $x \notin \Delta(a,S)$, *and* $x^{OPT} = a^{OPT}$, *then* $\text{dist}_{G'}((x,f) \le \text{dist}(x,a^{OPT})/(2-o(1))$.

*Proof.* $\text{dist}(x,a^{OPT}) \ge \text{dist}(x,a) - \text{dist}(a,a^{OPT}) \ge \text{dist}(x,a) - 2\delta_{a,S}/\alpha$. On the other hand, by Claim 25.2, $\text{dist}_{G'}(x,f) \le (\text{dist}(x,a) + \text{dist}(a,a'))/2 \le \text{dist}(x,a)/2 + \delta_{a,S}/\alpha$. Since $\text{dist}(x,a) = \Omega(\delta_{a,S})$, the claim follows. ∎

In order to deal with $x \in \Delta(a, S)$, we first prove the next claim.

CLAIM 25.7. *If* $\operatorname{dist}(a, a^{OPT}) \le 2\delta_{a,S}/\alpha$ *and* $x \in \Delta(a, S)$, *then* $\operatorname{dist}(x, OPT) \ge \operatorname{dist}(x, a^{OPT})$.

*Proof.* If $x^{OPT} \ne a^{OPT}$, $x^{OPT} \notin \operatorname{cluster}(a, S)$, so $\operatorname{dist}(x, OPT) \ge (0.25+1/\alpha)\delta_{a,S}$. On the other hand, $\operatorname{dist}(x, a^{OPT}) \le \operatorname{dist}(x, a) + \operatorname{dist}(a, a^{OPT}) \le (0.25 - 1/\alpha + 2/\alpha)\delta_{a,S}$. □

CLAIM 25.8. *If* $\operatorname{dist}(a, a^{OPT}) \le 2\delta_{a,S}/\alpha$, *then* $\sum_{x\in\Delta(a,S)} \operatorname{dist}_{G'}(x, f) \le \sum_{x\in\Delta(a,S)} \operatorname{dist}(x, OPT)$.

*Proof.* By Claim 25.7, $\sum_{x\in\Delta(a,S)} \operatorname{dist}(x, OPT) \ge \sum_{x\in\Delta(a,S)} \operatorname{dist}(x, a^{OPT}) \ge$ 1-mediancost$_{\Delta(a,S)} \ge (\sum_{x\in\Delta(a,S)} \operatorname{dist}(x, a'))/(1 + 1/\alpha) \ge (2 - o(1)) \cdot \sum_{x\in\Delta(a,S)} \operatorname{dist}_{G'}(x, f)$. □

Thus, if $|OPT \cap \Delta(a, S)| = 1$, the reassignments decrease the cost, by Claims 25.4, 25.5, 25.6, and 25.8.

*Finally, suppose that* $|OPT \cap \operatorname{cluster}(a, S)| > 1$. By Claim 25.4 we can restrict our attention to $x \in \operatorname{cluster}(a, S)$. By Claim 25.3, $\sum_{x\in\operatorname{cluster}(a,S)} \operatorname{dist}_{G'}(x, f) < \operatorname{clustercost}(a, S)$. Since $a \notin LARGE$, this cost is bounded by $\operatorname{cost}(S)/(q\alpha^3 \log n)$. However, by Lemma 24, $OPT$ misses $\le 3\alpha^2 q$ facilities in $S$, and hence $OPT$ can make multiple hits of at most $(3\alpha^2 q - q)$ facilities in $S$, so our total cost encountered in this case is $o(\operatorname{cost}(S)/(\alpha \log n))$, as claimed. This completes the proof of Theorem 25. □

## 5.11. Eliminating facilities with small shiftcost.

THEOREM 26. *Assume that Greedy$(q, S)$ is not successful and that we are given a $k$-median with* $\operatorname{cost}(T) = O(k\text{-mediancost})$. *We can then free* $p \le p_{\max}$ *facilities from* $T \cap \bigcup_{a\in SMALLSHIFT} \Delta(a, S)$, *increasing the cost of $T$ by at most* $(2.5 - 2/\alpha)p\,\text{smallshiftcost} + o(\operatorname{cost}(T)/(\alpha \log n))$.

*Proof.* We will identify a set $D \subseteq SMALLSHIFT$, $|D| = p$, of facilities from $S$ that are all hit by $T$, and let the modified set be $T' = T \setminus \bigcup_{a\in D} \Delta(a, S)$. Then $|T'| \le |T| - p$, as desired.

For the analysis of the cost increase, for each $a \in D$, we will reassign all $x$ with $x^T \in \Delta(a, S)$ to $x^{T'} = b^T$, where $b$ is the nearest facility in $S$ to $a$. Hence $D$ needs to be selected so that $b^T \notin \Delta(c, S)$ for any $c \in D$. We will then prove Theorem 26 by showing that the reassignment cost over all $x$ with $x^T \in \Delta(a, S)$ is $(2.5 - 2/\alpha)\operatorname{shiftcost}(a, S) + o(\operatorname{cost}(T)/(\alpha p \log n))$.

More precisely, the set $D$ will be selected subject to the following conditions:

(i) $c \notin D$ if there is a $b \in S$ not hit by $T$ with $b^T \in \Delta(c, S)$. By Lemma 24, there are $\le 3\alpha^2 q$ facilities $b \in S$ not hit by $T$, so this prevents only $\le 3\alpha^2 q$ facilities from entering $D$.

(ii) $a \notin D$ if $\sum_{x^T\in\Delta(a,S), x\notin\operatorname{cluster}(b,x)} \operatorname{dist}(x, T) \ge 4\operatorname{cost}(T)/(\alpha^2 p \log n)$. For each $x$, there is at most one $a \in S$ with $x^T \in \Delta(a, S)$, and thus $\sum_{a\in S} \sum_{x^T\in\Delta(a,S), x\notin\operatorname{cluster}(a,S)} \operatorname{dist}(x, T) \le \operatorname{cost}(T)$. This prevents at most $\alpha^2 p \log n/4 \le \frac{3}{4}\alpha^4 q \log n$ facilities from entering $D$.

(iii) $b \notin D$ if $b$ is hit by $T$ and $b$ is nearest in $S$ to some $a \in D$.

Consider some $a \in D$ with $b$ nearest to $a$ in $S$. We want to argue that $b^T \in T'$, and hence that $b^T \notin \Delta(c, S)$ for any $c \in D$. If $b$ is not hit by $T$, this follows immediately from (i). However, if $b$ is hit, $b^T \in \Delta(b, S)$, and by (iii), $b \notin D$.

As mentioned above, we are going to reassign all $x$ with $x^T \in \Delta(a, S)$ to $b^T$. Then the cost increase for $x$ is bounded by $\operatorname{dist}(x^T, b^T) \le 2\operatorname{dist}(x^T, b) \le 2\,(\operatorname{dist}(x^T, a) +$

$\operatorname{dist}(a,b)) \leq (2.5 - 2/\alpha)\,\delta_{a,S}$. Hence, the cost increase over all $x \in \operatorname{cluster}(a,S)$ is at most $\sum_{x\in\operatorname{cluster}(a,S)}((2.5-2/\alpha)\,\delta_{a,S}) \leq (2.5-2/\alpha)\operatorname{shiftcost}(a,S)$.

Now, consider $x \notin \operatorname{cluster}(a,S)$. Then $\operatorname{dist}(x,x^T) \geq \delta_{a,S}/4$, so the cost increase is $\geq (2.5-2/\alpha)\,\delta_{a,S} > 10\operatorname{dist}(x,x^T)$. Hence by (ii), for each $a \in D$, the cost increase over all $x$ with $x^T \in \Delta(a,S)$ and $x \notin \operatorname{cluster}(a,x)$ is at most $40\operatorname{cost}(T)/(\alpha^2 p\log n) = o(\operatorname{cost}(T)/(\alpha p\log n)$.

Thus, the total reassignment cost over all $x$ with $x^T \in \Delta(a,S)$ is at most $(2.5 - 2/\alpha)\operatorname{shiftcost}(a,S) + o(\operatorname{cost}(T)/(\alpha p\log n))$, as desired.

Our last problem is to construct the set $D$. As mentioned above, (i) and (ii) rule out at most $3\alpha^2 q + \frac{3}{4}\alpha^4\log n$ facilities. Since $|SMALLSHIFT| = \alpha^4 q\log n$, this leaves us with a subset $A \subseteq SMALLSHIFT$ of $\Omega(\alpha^4 q\log n)$ facilities satisfying (i) and (ii). By Lemma 21, we can mark at least half the facilities in $A$ so that $a$ is not marked if it is nearest in $S$ to some marked facility $b$. Then (iii) is trivially satisfied by picking $D$ as any set of $p \leq p_{max} = 3\alpha^2 q = o(\alpha^4 q\log n)$ marked facilities from $A$. ∎

**5.12. The true recursion.** In order to deal formally with the loss factors mentioned at the end of section 5.9, we need to consider graphs of the type $G^{c(d)}$, denoting the graph obtained from $G$ by multiplying the lengths of edges from free facilities by $c$ if they are to nonsingles and by $d$ if they are to singles. Also, define $G^c = G^{c(c)}$.

Our recursive solution $S$ to the $k$-median problem for $G$ with free facilities and singles will, with probability $(1 - O(1/n))$, satisfy

$$\operatorname{cost}_{G^{2(\phi)}}(S) \leq \left(1 + \frac{1}{(\alpha\log n)}\right)^{\log k}(1+\varepsilon)12 \times k\text{-mediancost}, \tag{3}$$

where $\phi = (2.5 - 2/\alpha)/(0.25 - 2.5/\alpha) = 10 + o(1)$ and $\varepsilon = o(1)$ is to be determined. Then the right-hand side reduces to $(12+o(1))k$-mediancost, and by definition, $\operatorname{cost}(S) \leq \operatorname{cost}_{G^{2(\phi)}}(S)$, so (3) implies $\operatorname{cost}(S) \leq (12 + o(1))k$-mediancost.

We now have to modify our basic recursion (cf. Algorithm G) so as to deal with free facilities and singletons so that we can formally prove (3).

As a very first preliminary step, if there are several free facilities in our input graph, we contract them into a single free facility $f_{in}$, where $\ell(f_{in}, x)$ is the shortest length of an edge from a free facility to $x$, and $\infty$ otherwise.

To implement step G.1, we modify the material from sections 5.2 and 5.4 to prove the next result.

Theorem 27. *In $\tilde{O}(m)$ time, with probability $1 - O(1/n)$, we can construct a $k + k/\log^2 n$-median $S$ with* $\operatorname{cost}_{G^{2(\phi)}}(S) \leq (12 + o(1)) \times k$-mediancost.

*Proof.* Concerning the sampling in section 5.2, we simply start Algorithm D by setting $S := \{f_{in}\}$. The algorithm is then run as before except that distances are computed in $G^2$. For the analysis, we consider all points assigned $f_{in}$ to be happy from the beginning. We then preserve that $\operatorname{dist}_{G^2}(x,S) \leq 2\operatorname{dist}(x,OPT)$ for all happy $x$. Consequently, the final sampled $F$ will satisfy $k$-mediancost$^{\subseteq F}_{G^2} \leq (2+o(1))\operatorname{cost}(OPT)$, as in Theorem 11.

Taking $H = G^2$ as starting point for the techniques in section 5.4, the first step is to include all links from $f_{in}$ in the graph $H^\ell$ to which we apply the algorithm from [19]. For the facility location in [19, section 2], we set the cost of $f_{in}$ to 0, whereas the cost is $z$ for all other points. To prove Theorem 27, we need to show that singles can be included optimally. For a given $z$ it is optimal to include a single $s$ linked to free facility $f$ if and only if $z \geq \operatorname{dist}(s,f)$. Hence, we do not pay the factor 3 from [19, Lemma 6]. Also, we avoid paying the factor 2 for rounding in [19, sections 3.3–5] if we make sure that $A$ and $B$ pick the same free facilities. To achieve this with $z_1 \geq z_2$ as in [19, section 3.3], we note that we can freely choose whether to include $s$ if $\text{dist}(a^p, f) \in [z_2, z_1]$, and hence we just need to make the same choices for $A$ and $B$. Thus we avoid any penalty for singles. Consequently, with $k^\ell = k + k/\log^2 n$, we end up with a $k^\ell$-median $S$ for $H$ with $\text{cost}_{H^{1(\phi/2)}}(S) = (1+o(1)) \max\{6, \phi/2\} k\text{-mediancost}_H^{\subseteq F} = (6+o(1))k\text{-mediancost}_H^{\subseteq F}$. Since $H^{1(\phi/2)} = G^{2(\phi)}$, we get $\text{cost}_{G^{2(\phi)}}(S) = (2+o(1))\text{cost}_{H^{1(\phi/2)}}(S) \leq (12+o(1))k\text{-mediancost}$, as desired. $\square$

Concerning shiftcost, we define $\text{shiftcost}(f_{in}, S) = \infty$, coding that $f_{in}$ may not be eliminated. For each input single facility $s$, we define $\text{shiftcost}(s, S) = \text{dist}(s, f_{in})/(0.25 - 2.5/\alpha)$, thus reversing the effect of step G.4.4 from when $s$ was created. Step G.2 uses this definition of shiftcost for the ordering.

The application of $\text{Greedy}(q, S)$ in step G.3 is unchanged, and our success criterion is still that the cost increase is bounded by $\text{cost}(S)/\alpha$. Then $\text{cost}_{G^{2(\phi)}}(\text{Greedy}(q, S)) \leq (1+\phi/\alpha)\text{cost}_{G^{2(\phi)}}(S) \leq (12+\varepsilon)k\text{-mediancost}$ with $\varepsilon = o(1)$. This defines the $\varepsilon$ in (3), which is hence satisfied if Greedy is successful.

If Greedy was not successful, we need to argue that the analysis from section 5.8 is still valid. All measures are still done in $G$. The free facility $f_{in}$ does not cause any problems. For a single facility $s$, we have $\text{cluster}(s, S) = \Delta(s, S) = \{s\}$, $\text{cost}(s, S) = 0$, and $\text{shiftcost}(s, S) = \text{dist}(s, f_{in})/(0.25 - 2.5/\alpha)$, so $s$ is concentrated. The cost of eliminating $s$ is $\text{dist}(s, f_{in})$. Observation 23 still holds for nonsingles, and for a single $s$, we note that if $T$ misses $s$, $\sum_{x \in \text{cluster}(s,S)} \text{dist}(x, T) = \text{dist}(s, f_{in}) = (0.25 - 2.5/\alpha)\text{shiftcost}(s, S)$. This bound suffices for our application of Observation 23 in Lemma 24, giving us our $3\alpha^2 q$ bound on the maximal number of missed facilities.

In step G.4.1 we remove $f_{in}$ from $U$, and in step G.4.3 we only set $\ell(x, f) = \text{dist}(x, a')/2$ if $f_{in}$ is not on all shortest paths from $x$ to $a'$. Our purpose here is to avoid reducing previously reduced distances. To see what happens with a single $s \in U$, first note that step G.4.2 sets $s' = s$. Also, since $f_{in}$ is the only neighbor $s$, step G.4.3 has no effect. Finally, a new single $t \in AUX$ from step G.4.4 has $\text{dist}_{G'}(t, f) = (0.25 - 2.5/\alpha)\text{smallshiftcost} \leq \text{dist}(s, f_{in})$.

For step G.4, we need to establish that Theorem 25 remains true, that is there is a $k'$-median $OPT'$ for $G'$ with $\text{cost}_{G'}(OPT') \leq (1 + o(1/(\alpha \log n)))\text{cost}(OPT)$. The construction of $OPT'$ is unchanged. Our reasoning above about singles implies that the cost over $\text{cluster}(s, S) = \{s\}$ goes down from $\text{dist}(s, f_{in})$ to $\text{dist}(s^s, f) \leq \text{dist}(s, f_{in})$ if $s \notin OPT$, and otherwise it stays 0.

For a nonsingle $a \in U$, we need to argue that our analysis is preserved despite the constraint that we do not set $\ell(x, f) \leq \text{dist}(x, a')/2$ if all shortest paths from $x$ to $a'$ go through $f_{in}$. However, a point $x$ is only affected by changes around $a$ if $x \in \Delta(a, S)$ or $x^{OPT} \in \Delta(a, S)$. Hence our analysis is preserved if we can prove the next results.

LEMMA 28. *If $x \in \Delta(a, S)$ or $x^{OPT} \in \Delta(a, S)$, no shortest path from $x$ to $a'$ passes $f_{in}$.*

*Proof.* By Claim 25.2, $\text{dist}(a, a') \leq 2\delta_{a,S}/\alpha$.

Now, if $x \in \Delta(a, S)$, $\text{dist}(x, a') \leq (0.25 - 1/\alpha)\delta_{a,S} + 2\delta_{a,S}/\alpha = (0.25 + o(1))\delta_{a,S}$, whereas $\text{dist}(x, f_{in}) \geq 0.75\,\delta_a$, so clearly $f_{in}$ cannot be on a shortest path from $x$ to $a'$.

Suppose $x^{OPT} \in \Delta(a, S)$ and that a shortest path from $x$ to $a'$ passs $f_{in}$. Then $\text{dist}(x, x^{OPT}) \geq \text{dist}(x, a') - (\text{dist}(a', a) + \text{dist}(a, x^{OPT}) \geq \text{dist}(x, a') - (2/\alpha + 0.25 - 1/\alpha)\delta_{a,S} = \text{dist}(x, a') - (0.25 + o(1))\delta_{a,S}$. However, $\text{dist}(x, f_{in}) \leq \text{dist}(x, a') - \text{dist}(a', f_{in}) \leq \text{dist}(x, a') - (\text{dist}(a, f_{in}) - \text{dist}(a, a')) \leq \text{dist}(x, a') - (1 - 2/\alpha)\delta_{a,S}$.

Since $f_{in} \in OPT$, this contradicts $x^{OPT}$ nearest to $x$ in $OPT$. ◻

Thus Theorem 25 remains true, and $k'$-mediancost$(G') \le (1 + o(1/(\alpha \log n))) \times$ $k$-mediancost$(G)$.

We now apply the recursive step G.5. Using (3) inductively, we get a $k'$-median $S'$ for $G'$ with $\mathrm{cost}_{G'^{2(\phi)}}(S') \le (1 + 1/(\alpha \log n))^{\log k'}(1 + \varepsilon)12 \times k\text{-mediancost}(G')$, which since $k' = o(k) < k/4$, is bounded by $(1 + 1/(\alpha \log n))^{(\log k)-1}(1 + \varepsilon)12 \times$ $k$-mediancost$(G)$.

Concerning step G.6.1, we want to argue as follows.

LEMMA 29. $\mathrm{cost}_{G^{2(\phi)}}(S'') \le \mathrm{cost}_{G'^{2(\phi)}}(S') - p(2.5 - 2/\alpha)\,\mathrm{smallshiftcost}$.

*Proof.* First note that when we delete $AUX$ from $G'$ and $S'$, for each of the $p$ facilities in $S' \setminus AUX$, we save $\phi(0.25 - 2.5/\alpha)$smallshiftcost $> (2.5 - 2/\alpha)$ smallshiftcost, as desired.

It remains to show that the deletion of $f_{in}$ with incident edges and the inclusion of $\{a'\}_{a \in U}$ does not increase our cost. Consider any $x$. The assignment cost of $x$ is affected only if either (a) $x^{S'} = f$ or (b) a shortest path from $x$ to $x^{S'}$ uses an edge incident to $f$. However, the two cases coincide, for clearly (a) implies (b), and (b) implies that $f$ is as near to $x$ as $x^{S'}$, and hence we can assume (a).

By (a), $x$ is not a single, since all singles linked to $f$ have been deleted. On a shortest path from $x$ to $f$ in $G'$ it is only the last edge $(y, f)$ that is incident to $f$. Then the assignment cost for $x$ was $\mathrm{dist}_{G'^{2(\phi)}}(x, y) + 2\ell(y, f)$, and $\mathrm{dist}_{G'^{2(\phi)}}(x, y) = \mathrm{dist}_{G^{2(\phi)}}(x, y)$. If $x$ is now assigned to $a'$, we know that $\ell(y, f) = \mathrm{dist}(y, f)/2$. Further, the edge $(y, f)$ was created only because a shortest path from $x$ to $a'$ did not contain $f$, and hence $\mathrm{dist}(y, f) = \mathrm{dist}_{G^{2(\phi)}}(y, f)$. Thus, the assignment cost for $x$ is unchanged. ◻

For the final step G.6.2, we want to prove the following version of Theorem 26 dealing with free facilities and singles.

THEOREM 30. *Assume that Greedy$(q, S)$ is not successful and that we are given a $k$-median with* $\mathrm{cost}(T) = O(k\text{-mediancost})$. *We can then free $p \le p_{\max}$ facilities from $T \cap \bigcup_{a \in SMALLSHIFT} \Delta(a, S)$, increasing the cost of $T$ in $G^{2(\phi)}$ by at most* $(2.5 - 2/\alpha)p\,\mathrm{smallshiftcost} + o(\mathrm{cost}(T)/(\alpha \log n))$.

*Proof.* Our proof consists of small modifications to the proof of Theorem 26. In particular, we will try to avoid the high cost of reassigning points to $f_{in}$ in $G^{2(\phi)}$.

As in Theorem 26, we will identify a set $D \subseteq SMALLSHIFT$, $|D| = p$, of facilities from $S$ that are all hit by $T$, and let the modified set be $T' = T \setminus \bigcup_{a \in D} \Delta(a, S)$.

Let $a \in D$, and let $b$ be the nearest facility to $a$ in $S$. If $b = f_{in}$, we will reassign all $x$ with $x^T \in \Delta(a, S)$ to $b^T = f_{in}$, as in Theorem 26. However, if $b \ne f_{in}$, we will reassign all $x$ with $x^T \in \Delta(a, S)$ to $b^{T \setminus \{f_{in}\}}$. For the latter case, we need to make sure that $b^{T \setminus \{f_{in}\}}$ is not deleted. This is done simply by changing (i) to

(i)$'$ $c \notin D$ if there is a $b \in S$ not hit by $T$ with $b^{T \setminus \{f_{in}\}}$.

Consider some $a \in D$ with $b$ nearest to $a$ in $S$. If $b \ne f_{in}$, we need to argue that $b^{T \setminus \{f_{in}\}} \in T'$, and hence that $b^{T \setminus \{f_{in}\}} \notin \Delta(c, S)$ for any $c \in D$. If $b$ is not hit by $T$, this follows immediately from (i)$'$. However, if $b$ is hit, then $b^{T \setminus \{f_{in}\}} \in \Delta(b, S)$, and by (iii), $b \notin D$.

Concerning the reassignment costs for points $x$ with $x^T \in \Delta(a, S)$, if $a \in D$ is single, the only such point is $a$ itself, and $a$ will be reassigned to $a^{T'} = f_{in}$ at cost $\mathrm{dist}_{G^{2(\phi)}}(a, f_{in}) = \phi\,\mathrm{dist}(a, f_{in}) = \phi(0.25 - 2.5/\alpha)\,\mathrm{shiftcost}(a, S) = (2.5 - 2/\alpha)\,\mathrm{smallshiftcost}$.

Now consider a nonsingle $a \in D$, and let $x \in \Delta(a, S)$. As in the analysis from Theorem 26, we want to argue that the reassignment cost is at most $2\,\mathrm{dist}(x^T, b)$, where

$b$ is nearest to $a$ in $S$. If $b = f_{in}$, $x$ is reassigned to $b$ at cost $\leq \text{dist}_{G^{2(\phi)}}\text{dist}(x^T, b) \leq 2\,\text{dist}(x^T, b)$, as desired. However, if $b \neq f_{in}$, $\text{dist}(b, b^{T\setminus\{f_{in}\}}) \leq \text{dist}(b, x^T)$ since $x^T \neq f_{in}$. Hence the reassignment cost for $b \neq f_{in}$ is $\leq \text{dist}(x^T, b^{T\setminus\{f_{in}\}}) \leq 2\,\text{dist}(x^T, b)$.

Having established that the reassignment cost is at most $2\,\text{dist}(x^T, b)$, we can directly apply the rest of the analysis from Theorem 26, thus showing that our total cost increase over all $x$ with $x^T \in \Delta(a, S)$ is $\leq (2.5 - 2/\alpha)\,\text{shiftcost}(a, S) + o(\text{cost}(T)/(\alpha p \log n)$, as desired.

The set $D$ is picked as in Theorem 26, noting that singles can always be added to $D$ since they are never nearest to any other facility than $f_{in}$. ∎

By Lemma 29 and Theorem 30—with $T = S''$ and $S^*$ the result of the modification—$\text{cost}_{G^{2(\phi)}}(S^*) \leq o(\text{cost}(S'')/(\alpha \log n)) + \text{cost}_{G'^{2(\phi)}}(S')$, where $\text{cost}(S'') < \text{cost}_{G'^{2(\phi)}}(S')$. Further we saw that $\text{cost}_{G'^{2(\phi)}}(S') \leq (1+1/(\alpha \log n))^{(\log k)-1}(1+\varepsilon)12\times k\text{-mediancost}(G)$, so we conclude that $\text{cost}_{G^{2(\phi)}}(S^*) \leq (1+1/(\alpha \log n))^{\log k}(1+\varepsilon)12 \times k\text{-mediancost}(G)$, thus establishing (3).

As mentioned just below the introduction of (3), it directly implies our main result, which is the following.

THEOREM 31. *W.h.p., in $\tilde{O}(m)$ time and space, we can find a $k$-median solution $S$ with* $\text{cost}(S) \leq (12 + o(1))k\text{-mediancost}$.

## 5.13. The *k*-median for other metrics.

So far, we have considered solving the $k$-median problem only for graphs, but our technique is really much more general. The only way we use the input graph is to compute all points nearest point in sets $F$ of potential facilities; otherwise, we work internally on auxiliary graphs of size $\tilde{O}(n)$. Thus, we have really proved the following more general result.

THEOREM 32. *W.h.p., using a polylogarithmic number of all points nearest point in sets of facilities $F \subseteq P$ of size $\tilde{O}(k)$ as well as $\tilde{O}(n)$ internal computation time and space, we can find a $k$-median solution $S$ with* $\text{cost}(S) \leq (12 + o(1))k\text{-mediancost}$.

Clearly Theorem 32 implies Theorem 31 since an all points nearest marked neighbor computation on a graph can be done in linear time using a single source shortest path computation (cf. Observation 1). However, with a distance oracle, the marked neighbor computation is trivially done by comparing the distances from each $x \in P$ to each $f \in F$, hence with $\tilde{O}(kn)$ distance queries total. Thus Theorem 32 implies our $\tilde{O}(kn)$ time bound with linear space for general distance oracles, as stated earlier in Proposition 12. However, in, say, Hamming space, we can do better using approximate nearest neighbor queries.

COROLLARY 33. *For points in Hamming space of arbitrary dimension, for $0 < \varepsilon \leq 1$, w.h.p., we can solve the $k$-median problem within a factor $(12 + o(1))/\varepsilon$ in $\tilde{O}(nk^{\varepsilon})$ time.*

*Proof.* For $0 < \varepsilon \leq 1$, Indyk and Motwani [17], [16, p. 22] have shown that we can in $\tilde{O}(|F|^{1+\varepsilon})$ time and space build a data structure that for an arbitrary point finds its $1/\varepsilon$-approximate nearest neighbor in $\tilde{O}(F|^{\varepsilon})$ time with constant probability. The dimension of the space does not matter, as long as we can access each coordinate of a point in constant time. Using $O(\log n)$ such structures, all points in $P$ will find an $(1/\varepsilon)$-approximate nearest neighbor $x^F$ in $F$ w.h.p. Our total running time is then $\tilde{O}(nk^{\varepsilon})$. With these $(1/\varepsilon)$-approximate nearest neighbors, we can perform at least as well as with exact neighbors with all distances multiplied by $(1/\varepsilon)$, so our approximation factor becomes $(12 + o(1))/\varepsilon$. ∎

Since $\varepsilon = \Theta(1)$, our Hamming space approximation factor is constant, and for $\varepsilon < 1$, we beat $\Omega(kn)$ lower-bound on the time needed for any approximation factor in the general distance oracle model.

**Appendix. 1-median.** We need a result from [18] on approximating 1-medians. Since [18] may never be published, we present here the relevant result. We prove that we can find a factor $(1+\varepsilon)$ approximate 1-median by computing all distances from $O((\log n)/\varepsilon^2)$ random points as follows.

THEOREM 34. *Let $\varepsilon \leq 1$. Let $Q$ be a random multisubset of $P$. Let $a \in P$ minimize $\sum_{x\in Q} \mathrm{dist}(a,x)$. Then* $\Pr(\mathrm{cost}(\{a\}) \geq (1+\varepsilon) \times \text{1-mediancost}) \leq ne^{-\varepsilon^2|Q|/16}$.

*Proof.* Let *opt* be the optimal 1-median for $P$, and let $b$ be an arbitrary point in $P$ with $\mathrm{cost}(\{b\}) > (1+\varepsilon)\mathrm{cost}(\{opt\})$. There are at most $n-1$ such choices of $b$, and thus the theorem follows if

$$\Pr\left(\sum_{x\in Q} \mathrm{dist}(opt,x) \geq \sum_{x\in Q} \mathrm{dist}(b,x)\right) < e^{-\varepsilon^2|Q|/64}. \tag{4}$$

To prove (4), we study the random variable $X = \sum_{x\in Q} \frac{\mathrm{dist}(b,x)-\mathrm{dist}(opt,x)+\mathrm{dist}(opt,b)}{2\mathrm{dist}(opt,b)}$. By the triangle inequality, this is the sum of random variables between 0 and 1. Our bad event is that $X \leq |Q|/2$. On the other hand,

$$E(X) = \frac{|Q|}{|P|}\sum_{x\in P} \frac{\mathrm{dist}(b,x)-\mathrm{dist}(opt,x)+\mathrm{dist}(opt,b)}{2\mathrm{dist}(opt,b)}.$$

Since $\mathrm{dist}(b,x)+\mathrm{dist}(opt,x) \geq \mathrm{dist}(opt,b)$ and $\sum_{x\in P}\mathrm{dist}(x,b) > (1+\varepsilon)\sum_{x\in P}\mathrm{dist}(x,opt)$, $\sum_{x\in P}(\mathrm{dist}(b,x)-\mathrm{dist}(opt,x)) \geq \sum_{x\in P}\frac{\varepsilon}{2+\varepsilon}\mathrm{dist}(opt,b)$, so $E(X) \geq |Q|(1+\frac{\varepsilon}{2+\varepsilon})/2$. Consequently, $|Q|/2 \leq (1-\delta)E(X)$ with $\delta = \frac{\varepsilon}{2+\varepsilon}/(1+\frac{\varepsilon}{2+\varepsilon}) \geq \varepsilon/4$ for $\varepsilon \leq 1$. Applying a Chernoff bound from [24, Theorem 4.2], $\Pr(X \leq |Q/2|) \leq \Pr(X \leq (1-\delta)E(X)) < e^{-\delta^2 E(X)/2} < e^{-\varepsilon^2|Q|/64}$. This completes the proof of (4). □

**Acknowledgment.** I would like to thank the referees from *SIAM J. Computing* for many good suggestions concerning the presentation of this paper.

## REFERENCES

[1] M. CHARIKAR AND S. GUHA, *Improved combinatorial algorithms for the facility location and k-median problems*, in Proceedings of the 40th Annual Symposium on Foundations of Computer Science, New York, 1999, IEEE Press, Piscataway, NJ, pp. 378–388.

[2] M. CHARIKAR, S. GUHA, E. TARDOS, AND D. B. SHMOYS, *A constant-factor approximation algorithm for the k-median problem*, J. Comput. System Sci., 65 (2002), pp. 129–149.

[3] E. COHEN, *Size-estimation framework with applications to transitive closure and reachability*, J. Comput. System Sci., 55 (1997), pp. 441–453.

[4] E. COHEN AND U. ZWICK, *All-pairs small-stretch paths*, J. Algorithms, 38 (2001), pp. 335–353.

[5] R. COLE, R. HARIHARAN, M. LEWENSTEIN, AND E. PORAT, *A faster implementation of the Goemans–Williamson clustering technique*, in Proceedings of the 12th Annual ACM-SIAM Symposium on Discrete Algorithms, Washington, DC, 2001, SIAM, Philadelphia, 2001, pp. 17–25.

[6] T. H. CORMEN, C. E. LEISERSON, AND R. L. RIVEST, *Introduction to Algorithms*, MIT Press/McGraw-Hill, Cambridge, MA, New York, 1990.

[7] M. R. GAREY AND D. S. JOHNSON, *Computer and Intractability: A Guide to NP-Completeness*, W. H. Freeman, San Francisco, 1979.

[8] A. GOEL, P. INDYK, AND K. VARADARAJAN, *Reductions among high dimensional proximity problems*, in Proceedings of the 12th Annual ACM-SIAM Symposium on Discrete Algorithms, Washington, DC, 2001, SIAM, Philadelphia, 2001, pp. 769–778.

[9] T. F. GONZALES, *Clustering to minimize the maximum intercluster distance*, Theoret. Comput. Sci., 38 (1985), pp. 293–550.

[10] S. GUHA, *Approximation Algorithms for Facility Location Problems*, Ph.D. thesis, Computer Science Department, Stanford University, Stanford, CA, 2000.

[11] S. Guha and S. Khuller, *Greedy strikes back: Improved facility location algorithms*, J. Algorithms, 31 (1999), pp. 228–248.

[12] S. Guha, M. Mishra, R. Motwani, and L. O'Callaghan, *Clustering data streams*, in Proceedings of the 41st Annual Symposium on Foundations of Computer Science, Redondo Beach, CA, 2000, IEEE Press, Piscataway, NJ, pp. 359–366.

[13] D. Hochbaum and D. B. Shmoys, *A unified approach to approximation algorithms for bottleneck problems*, J. ACM, 33 (1986), pp. 533–550.

[14] W. L. Hsu and G. L. Nemhauser, *Easy and hard bottleneck problems*, Discrete Appl. Math., 1 (1979), pp. 209–216.

[15] P. Indyk, *Sublinear time algorithms for metric space problems*, in Proceedings of the 31st Annual ACM Symposium on Theory of Computing, Atlanta, GA, 1999, ACM, New York, pp. 428–434.

[16] P. Indyk, *High-Dimensional Computational Geometry*, Ph.D. thesis, Computer Science Department, Stanford University, Stanford, CA, 2000.

[17] P. Indyk and R. Motwani, *Approximate nearest neighbors: Towards removing the curse of dimensionality*, in Proceedings of the 30th Annual ACM Symposium on Theory of Computing, Dallas, TX, 1998, ACM, New York, pp. 604–613.

[18] P. Indyk and M. Thorup, *Approximate* 1*-Medians*, manuscript, 2000.

[19] K. Jain and V. V. Vazirani, *Approximation algorithms for metric facility location and k-median problems using primal-dual and Lagrangian relaxation*, J. ACM, 48 (2001), pp. 274–296.

[20] S. Jamin, C. Jin, Y. Jin, D. Raz, Y. Shavitt, and L. Zhang, *On the placement of Internet instrumentations*, in Proceedings of the 19th INFOCOM meeting, Tel-Aviv, Israel, 2000, IEEE Press, Piscataway, NJ, pp. 26–30.

[21] B. Li, M. Golin, G. G. Italiano, X. Deng, and K. Sohraby, *On the optimal placement of web proxies in the Internet*, in Proceedings of the 18th INFOCOM, New York, 1999, IEEE Press, Piscataway, NJ, pp. 1282–1290.

[22] M. Luby, *A simple parallel algorithm for the maximal independent set problem*, SIAM J. Comput., 15 (1986), pp. 1036–1053.

[23] R. R. Mettu and C. G. Plaxton, *The online median problem*, SIAM J. Comput., 32 (2003), pp. 816–832.

[24] R. Motwani and P. Raghavan, *Randomized Algorithms*, Cambridge University Press, New York, 1995.

[25] L. Qiu, V. N. Padmanabhan, and G. M. Voelker, *On the optimal placement of web server replicas*, in Proceedings of the 20th INFOCOM, Anchorage, AK, 2001, IEEE Press, Piscataway, NJ, pp. 1587–1596.

[26] B. C. Tansel, R. L. Francis, and T. J. Lowe, *Location on networks: A survey. Parts* 1 *and* 2, Management Sci., 29 (1983), pp. 482–511.

[27] M. Thorup, *Quick k-median, k-center, and facility location for sparse graphs*, in Proceedings of the 28th International Colloquium on Automata, Languages, and Programming, Lecture Notes in Comput. Sci. 2076, 2001, Springer-Verlag, New York, pp. 249–260.

[28] M. Thorup, *Quick and good facility location*, in Proceedings of the 14th Annual ACM-SIAM Symposium on Discrete Algorithms, Baltimore, MD, 2003, SIAM, Philadelphia, 2003, pp. 178–185.

[29] M. Thorup and U. Zwick, *Approximate distance oracles*, in Proceedings of the 33rd Annual ACM Symposium on Theory of Computing, Crete, Greece, 2001, ACM, New York, pp. 183–192.

SIAM J. COMPUT.
Vol. 34, No. 2, pp. 433–452

# ONLINE SCHEDULING TO MINIMIZE AVERAGE STRETCH[*]

S. MUTHUKRISHNAN[†], RAJMOHAN RAJARAMAN[‡], ANTHONY SHAHEEN[§], AND JOHANNES E. GEHRKE[¶]

**Abstract.** We consider the classical problem of online job scheduling on uniprocessor and multiprocessor machines. For a given job, we measure the quality of service provided by an algorithm by the *stretch* of the job, which is defined as the ratio of the amount of time that the job spends in the system to the processing time of the job. For a given sequence of jobs, we measure the performance of an algorithm by the *average stretch* achieved by the algorithm over all the jobs in the sequence. The average stretch metric has been used to evaluate the performance of scheduling algorithms in many applications arising in databases, networks, and systems.

The main contribution of this paper is to show that the *shortest remaining processing time* (SRPT) algorithm is $O(1)$-competitive with respect to average stretch for both uniprocessors and multiprocessors. For uniprocessors, we prove that SRPT is 2-competitive; we also establish an essentially matching lower bound on the competitive ratio of SRPT. For multiprocessors, we show that the competitive ratio of SRPT is at most $9 + 2\sqrt{6} \leq 14$. Furthermore, we establish constant-factor lower bounds on the competitive ratio of any online algorithm for both uniprocessors and multiprocessors.

**Key words.** scheduling, online algorithms, competitive ratio, shortest remaining processing time (SRPT), stretch, slowdown

**AMS subject classifications.** 68W40, 68W10, 68Q99

**DOI.** 10.1137/S0097539701387544

## 1. Introduction.

Many servers, such as web and database servers, receive a continuous stream of requests with processing times that vary over several orders of magnitude. The main challenge in such a scenario is to schedule the stream of requests so as to provide various service guarantees such as response time, throughput, and fairness. In the combinatorial scheduling literature, this problem is typically abstracted as one of optimizing a suitably defined performance measure. Two classical performance measures are the *makespan*, which is the maximum completion time of any job, and the *total completion time*, which is the sum of the completion times of all the jobs; these are suitable for applications where the jobs are executed in batches. For jobs arriving continuously, a relevant parameter is the time that a job spends in the system, namely, the *response time* (also sometimes referred to as the *flow time*) of the job, which can be defined as the difference between the completion time and release time of the job. For over two decades, the *average response time* has been a popular performance measure in online scheduling research.

Recently, alternative performance measures have been considered. The premise

[*]Received by the editors April 7, 2001; accepted for publication (in revised form) May 26, 2004; published electronically February 3, 2005.
http://www.siam.org/journals/sicomp/34-2/38754.html

[†]Department of Computer Science, Rutgers University, Piscataway, NJ 08854 (muthu@cs.rutgers.edu). Part of this work was done while the author was at AT&T Shannon Laboratories, Florham Park, NJ 07932.

[‡]College of Computer and Information Science, Northeastern University, Boston MA 02115 (rraj@ccs.neu.edu). Part of this work was done when the author was at DIMACS, Piscataway, NJ.

[§]Loyola Marymount University, Los Angeles, CA 90045. Part of this work was done while the author was at DIMACS as part of the Research Experiences for Undergraduates program.

[¶]Department of Computer Science, Cornell University, Ithaca, NY 14853 (johannes@cs.cornell.edu).

underlying these studies is that an important parameter of performance for a job is the *factor by which it is slowed down relative to the time it takes on a unloaded system*. Formally, the *stretch* (also known as the *slowdown*) of a job is the ratio of the response time of the job to the processing time of the job. Intuitively, the stretch measure relates the users' waiting times to their demands, since it measures the quality of service provided to a job in proportion to its demand. It may also reflect users' psychological expectation that, in a system with highly variable job sizes, users are willing to wait longer for larger requests. Thus it is a reasonably fair measure of the service that an individual job gets in the system.

The stretch measure, more precisely, the *average stretch* (or equivalently, the *total stretch*), has been used to empirically study the performance of several applications, e.g., in databases [19], operating systems [15], and parallel systems [13]. Average stretch is used for measuring the performance of scheduling algorithms in the context of web servers or clusters thereof [14, 23], which process requests for downloading files of different sizes and types. The average stretch is also a good indicator of the total load of the system. A schedule with large average stretch is likely to have a majority of jobs waiting in queue for large periods of time, while a schedule may have small average queue size and yet a large average response time simply because one of the jobs has a large processing time.

To summarize, average stretch is a natural measure that fits many applications, and is a better indicator of system performance than the traditional average response time measure. Surprisingly, very little is known about scheduling algorithms that optimize (minimize) average stretch; in fact, the rather basic problem of minimizing average stretch on a single processor has not been studied.

In this paper, we study the problem of online scheduling to minimize average stretch on both single and multiprocessor systems. We focus on the *preemptive clairvoyant* version of the scheduling problem: by preemptive, we mean that jobs may be stopped before their completion and resumed later after processing other jobs in the interim; by clairvoyant, we mean that the processing time of each job is known at the time of its arrival. The aforementioned application of web server scheduling is an important example where preemptive clairvoyant schedules are useful.

Our main contribution is to show that the *shortest remaining processing time* (SRPT) heuristic is $O(1)$-competitive with respect to average stretch for both uniprocessors and multiprocessors. Our results show that the problem of optimizing the average stretch is fundamentally different from that of optimizing the average response time in the uniprocessor as well as in the multiprocessor case, although, interestingly, the difference is contrasting. (See section 1.2.) Our results, taken together with previous work on analyzing average response time [3, 18], imply that SRPT simultaneously optimizes the average response time and average stretch measures, up to constant factors, for both uniprocessors and multiprocessors.

**1.1. Problem and definitions.** We consider the following online scheduling scenario. A sequence of jobs arrives online, and the processing time of each job is known at the time of its arrival. Our goal is to produce a schedule to process the continuously arriving stream of such jobs. The quality of service we consider is the *stretch* of a job, which is defined as the ratio of the amount of time that the job spends in the system to the processing time of the job. Thus, if a job $j$ with processing time $p(j)$ arrives at time $r(j)$ and is completed at time $C(j)$, then the stretch of the job is given by $s(j) = (C(j) - r(j))/p(j)$. We seek to minimize the total stretch of the jobs in a given instance.

We study the problem of minimizing total stretch for both uniprocessor as well as multiprocessor scheduling. In the case of multiprocessor scheduling, we assume that all of the processors are identical. Thus, in the standard 3-field scheduling notation [17], the problems that we study may be listed as $1 \mid pmtn, r_j \mid \sum_j (C_j - r_j)/p_j$ and $Pm \mid pmtn, r_j \mid \sum_j (C_j - r_j)/p_j$, respectively.

Since our focus is on online algorithms, we perform a competitive analysis [22] of the algorithms under consideration. An online algorithm $\mathcal{A}$ is *r-competitive* if for every instance $\mathcal{J}$, $S(\mathcal{J})$ is at most $rS^*(\mathcal{J})$, where $S(\mathcal{J})$ and $S^*(\mathcal{J})$ represent the total stretch achieved by $\mathcal{A}$ and an optimal offline algorithm, respectively, on instance $\mathcal{J}$. The *competitive ratio* of an algorithm $\mathcal{A}$ is the infimum over all $r$ such that $\mathcal{A}$ is $r$-competitive.

Much of our work concerns the analysis of the well-known SRPT algorithm. For uniprocessors, SRPT is simply defined as follows: In each time step, process a job with the shortest remaining processing time among all the unfinished jobs. For an $m$-processor machine each time step consists of the following operation: If there are at least $m$ unfinished jobs, a set of $m$ jobs that have the $m$ shortest remaining processing times are processed, where we break ties arbitrarily; otherwise, every unfinished job is processed. (Since the processors are assumed to be identical, the particular assignment of an unfinished job to a processor at any step is irrelevant.[1])

**1.2. Our results.** Our main results are threefold:

- *Uniprocessor* SRPT. We show that for any instance $\mathcal{J}$ on a uniprocessor machine, the total stretch $S(\mathcal{J})$ achieved by SRPT is at most $S^*(\mathcal{J}) + |\mathcal{J}|$; since $S^*(\mathcal{J}) \geq |\mathcal{J}|$, this implies that SRPT is 2-competitive. We also establish an essentially matching lower bound on SRPT for uniprocessors. These results appear in section 2.
  Our upper bound for SRPT makes it an attractive algorithm for uniprocessor scheduling since it is nearly optimal with respect to total stretch while it is known to be truly optimal with respect to average response time. Indeed, recent experimental and analytical studies on scheduling algorithms for web servers [4, 14] have shown that SRPT outperforms several other scheduling policies currently used in web servers on real-world workloads; the analyses in the preceding studies adopt a Poisson model for the job arrival process, and general as well as heavy-tailed distributions for job service times.
- *Multiprocessor* SRPT. For multiprocessors, we show that for any instance $\mathcal{J}$ the total stretch $S(\mathcal{J})$ achieved by SRPT is at most $3S^*(\mathcal{J}) + (6 + 2\sqrt{6})|\mathcal{J}|$, which implies an upper bound of $9 + 2\sqrt{6} \leq 14$ on the competitive ratio, independent of the number of processors $m \geq 2$. These results appear in section 3.
  Our constant-factor competitiveness result for multiprocessors provides a surprising contrast to the recent lower bound established on the competitive ratio of any online algorithm with respect to average response time. It is shown in [18] that the best competitive ratio achievable for average response time on an $m$-processor machine is $\Omega(\max\{\log P, \log(n/m)\})$, where $P$ is the ratio of the maximum to the minimum processing time of any job and $n$ is the number of jobs. In [18], it is also shown that the competitive ratio of SRPT with respect to average response time on multiprocessors is optimal to within a constant factor. This indicates that SRPT is a compelling algorithm for

[1]A job may be preempted at one processor and later resumed at a different processor.

multiprocessor scheduling as well since it matches the best competitive ratio up to constant factors for both total stretch as well as average response time.
- *Lower bounds.* We show that there exists no online algorithm that minimizes average stretch for every instance; specifically, we show that the competitive ratio of every online algorithm is at least 1.04. We also show a lower bound of at least 7/6 on the competitive ratio for minimizing total stretch on any $2m$-processor machine, for any positive integer $m$. These results appear in section 4.

The average response time and total stretch metrics have the following interesting contrasting characteristics. For uniprocessors, while average response time can be optimized by an on-line algorithm, the best competitive ratio achievable for total stretch is strictly greater than 1. On the other hand, for multiprocessors, while a constant-factor competitive online algorithm exists for total stretch, the best competitive ratio for average response time grows logarithmically with the number of jobs.

**1.3. Related work.** While average response time has been studied extensively in the literature [3, 16, 21], the analytical study of stretch as a performance measure was initiated only recently [6]. All of the results presented in [6], however, concern the metric of *maximum stretch* (that is, $\max_i S_i$) and do not yield any useful bounds for total stretch. Recently, max-stretch was also studied in the context of performing file transfers over a network with given bandwidth constraints on the underlying links [11].

Two measures closely related to total stretch are weighted completion time and weighted flow time, each of which associate a weight with each job $i$. If we set the weight of job $i$ to the reciprocal of flow time, completion time also optimizes total stretch. From the point of view of approximation, however, the two metrics are different. A randomized online algorithm that achieves a competitive ratio of 4/3 with respect to weighted completion time is given in [20]; this result, however, does not directly yield any useful upper bound on the competitive ratio for total stretch. In fact, it is easy to construct schedules that have a constant-factor approximation ratio with respect to weighted completion time, with $w_i = 1/p_i$, and yet have a linear approximation ratio with respect to total stretch. For other results on the online and offline complexity of weighted completion time, see [12].

The weighted flow time with weights given by the reciprocal of processing times is identical to the total stretch metric. The best known approximation result for weighted flow time is the recent approximation scheme of [8], which takes time super-polynomial, but subexponential, in the input size. In a subsequent study [9], it has been shown that a quasi-polynomial approximation scheme (quasi-PTAS) is achievable for weighted flow time when $\Delta$ and the ratio of the maximum weight to minimum weight are both polynomially bounded. When applied to the special case of the total stretch metric, the techniques of [9] yield a PTAS. A PTAS for total stretch is also given in [7].

For multiprocessors, the work most closely related to our study is that of [18], which is discussed in section 1.2. In recent work [1], a polynomial-time approximation scheme is derived for the weighted completion time problem on multiprocessors.

**1.4. Overview of our analyses.** Our analysis of SRPT relies on a careful comparison of the queue of unfinished jobs in SRPT with the corresponding queue associated with any other scheduling algorithm at every time step. For the case of uniprocessors, we derive a tight bound on the total stretch of SRPT by placing a separate bound, for *every* unfinished job at *every* time step, on the contribution of the job to the total stretch of SRPT. A key component behind this approach is to appropriately map the jobs in SRPT's queue to the jobs in the queue associated with an arbitrary scheduling algorithm at every time step. The particular mapping enables us to analyze the total stretch of SRPT time step by time step. The relevant property of this mapping is illustrated in Figure 1 of section 2 and formally established in Lemma 2.8.

Our analysis for multiprocessors is more involved. The difficulty in the analysis arises due to the following observation made in [18]: there exist multiprocessor scheduling instances that may force SRPT to have many more unfinished jobs at certain time steps than an alternative schedule has at that time. As a result, the contribution to the total stretch corresponding to such time steps may be disproportionately larger for SRPT than for the other schedule. We overcome this hurdle by showing the existence of an appropriate partial mapping from the jobs in SRPT's queue to the jobs in the queue associated with the other schedule. In addition to placing an upper bound on the contribution of the mapped jobs in each step, the particular mapping enables us to place an upper bound on the exact or amortized contribution of the unmapped jobs in SRPT's queue at each time step. The mapping is illustrated in Figure 2 of section 3 and formally established in Lemma 3.5.

## 2. Analysis of SRPT for uniprocessor scheduling.

In this section, we analyze the competitiveness of SRPT on uniprocessors with respect to total stretch. An instance $\mathcal{J}$ of our scheduling problem consists of a set of jobs with arbitrary release times and processing times. For job $j \in \mathcal{J}$, let $r(j)$ and $p(j)$ denote the release time and processing time, respectively, of $j$. The performance of a schedule is measured by the total stretch achieved. Recall that the total stretch of a schedule for a given instance is the sum, taken over all jobs $j \in \mathcal{J}$, of the stretch of $j$ under the schedule, where the stretch of a job is the ratio of the response time of the job to the processing time of the job. Let $S(\mathcal{J})$ and $S^*(\mathcal{J})$ denote the total stretch of SRPT and the optimal total stretch, respectively, for instance $\mathcal{J}$. The main result of this section is the following theorem that places an upper bound on the total stretch achieved by SRPT.

THEOREM 2.1. *For any instance $\mathcal{J}$, $S(\mathcal{J})$ is at most $S^*(\mathcal{J}) + |\mathcal{J}|$.*

The proof of Theorem 2.1 is given in section 2.1. In terms of competitive ratio, Theorem 2.1 implies the following result.

COROLLARY 2.2. SRPT *is 2-competitive with respect to average stretch.*

*Proof.* For any scheduling algorithm, the stretch of any job is at least 1. Therefore $S^*(\mathcal{J})$ is at least $|\mathcal{J}|$. It thus follows from Theorem 2.1 that $S(\mathcal{J}) \leq 2S^*(\mathcal{J})$. ∎

We also show an essentially matching lower bound in Theorem 2.3, which is proved in section 2.2.

THEOREM 2.3. *For any real $\varepsilon > 0$ and any positive integer $n \geq 3$, there exists an instance $\mathcal{J}$ with $n$ jobs such that $S(\mathcal{J})$ is at least $S^*(\mathcal{J}) + n - 1 - \varepsilon$.*

### 2.1. Upper bound.

In this section, we prove Theorem 2.1. We begin by introducing some notation to characterize the execution of SRPT on a given instance $\mathcal{J}$. For any job $j$ in $\mathcal{J}$ and time $t \geq r(j)$, let $\rho_t(j)$ denote the remaining processing time of job $j$ at time $t$ under SRPT. For any time $t$, let $Q_t$ denote the set of all jobs in $\mathcal{J}$ that have been released at some time less than or equal to $t$ and have not been completed by SRPT; that is, $Q_t$ is the set of all jobs $j$ for which $r(j) \leq t$ and $\rho_t(j) > 0$. We rank all of the jobs in $Q_t$ in nondecreasing order of their remaining processing times, breaking ties according to an arbitrary ordering of the job IDs whenever necessary. For $i \geq 1$, let $q_{t,i}$ denote the job of rank $i$ in $Q_t$; if the job of rank $i$ is not defined, we set $q_{t,i}$ to be a job with processing time $\infty$. We refer to a job as *active* (resp., *waiting*) at time $t$ if the rank of the job is equal to 1 (resp., greater than 1). Note that in time step $t$, SRPT processes one unit of the job that is active at that time. Let $U_t$ denote the set of waiting jobs at time $t$.

We now express the total stretch $S(\mathcal{J})$ in SRPT as a sum, taken over all time steps $t$ and over all jobs in $Q_t$, of the reciprocal of the processing time of the job:

$$S(\mathcal{J}) = \sum_{j \in \mathcal{J}} \sum_{t : j \in Q_t} \frac{1}{p(j)} = \sum_{t} \sum_{j \in Q_t} \frac{1}{p(j)}.$$

We now introduce two new variables for notational convenience. Let $\alpha_t$ denote the sum of the reciprocals of the processing times of the jobs in $Q_t$. Thus, the total stretch $S(\mathcal{J})$ is simply $\sum_t \alpha_t$. Finally, let $\beta_t$ denote the sum of the reciprocals of the remaining processing times of the jobs in $Q_t$. Since the remaining processing time of a job at any time is at most its processing time, we obtain that $\alpha_t \leq \beta_t$ for all $t$.

We split the total stretch $S(\mathcal{J})$ into two parts $S_1(\mathcal{J})$ and $S_2(\mathcal{J})$, which represent the total contribution of the active jobs and the waiting jobs, respectively, in SRPT:

$$S_1(\mathcal{J}) = \sum_t \frac{1}{p(q_{t,1})} \quad \text{and}$$
$$S_2(\mathcal{J}) = \sum_t \sum_{j \in U_t} \frac{1}{p(j)}.$$

The total contribution of the active jobs can be calculated easily. Whenever a job $j$ is active, it contributes $1/p(j)$ to the term $S_1(\mathcal{J})$. Since an active job is always processed by SRPT, there are exactly $p(j)$ time steps when $j$ is active. Thus, $j$ contributes $p(j)/p(j) = 1$ to $S_1(\mathcal{J})$. We thus obtain the following lemma.

LEMMA 2.4. *For any instance $\mathcal{J}$, $S_1(\mathcal{J}) = |\mathcal{J}|$.* □

We place an upper bound on the contribution of the waiting jobs by means of a careful comparison of SRPT's queue $Q_t$ with the corresponding queue of an arbitrary scheduling algorithm $\mathcal{A}$. Just as we have characterized the execution of SRPT for a given instance $\mathcal{J}$ using the variables $\rho_t$, $Q_t$, and $q_t$, we can characterize the execution of $\mathcal{A}$ for $\mathcal{J}$. Let $\widetilde{Q}_t$, $\widetilde{\rho}_t$, $\widetilde{\alpha}_t$, and $\widetilde{\beta}_t$ denote the equivalent of $Q_t$, $\rho_t$, $\alpha_t$, and $\beta_t$, respectively, for the execution of $\mathcal{A}$. Furthermore, whenever we henceforth introduce a new notation for a variable associated with SRPT, it is implicit that the "tilde" version of the notation denotes the corresponding variable for $\mathcal{A}$.

We obtain the desired upper bound on $S_2(\mathcal{J})$ by showing the following key lemma that relates $\beta_t$ and $\widetilde{\alpha}_t$.

LEMMA 2.5. *For any time $t$, we have $\beta_t \leq \widetilde{\alpha}_t + \frac{1}{\rho_t(q_{t,1})}$.*

COROLLARY 2.6. *For any instance $\mathcal{J}$, $S_2(\mathcal{J}) \leq S^*(\mathcal{J})$.*

*Proof.* Since $1/p(j)$ is at most $1/\rho_t(j)$ for any job $j$, it follows that

$$S_2(\mathcal{J}) \leq \sum_t \left( \beta_t - \frac{1}{\rho_t(q_{t,1})} \right)$$
$$\leq \sum_t \widetilde{\alpha}_t \leq \widetilde{S}(\mathcal{J}),$$

where the last step is obtained by invoking Lemma 2.5. Since the above inequality holds for every scheduling algorithm $\mathcal{A}$, the desired claim follows. □

Theorem 2.1 follows directly from Lemma 2.4 and Corollary 2.6. Thus all that remains to prove is Lemma 2.5. A crucial component of our proof of Lemma 2.5 is

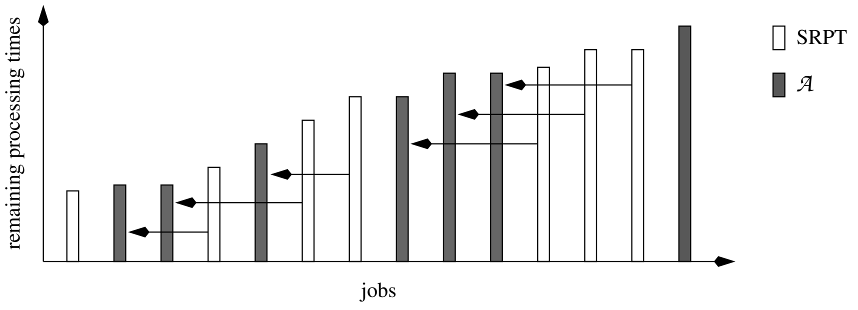


FIG. 1. *The remaining processing times of the jobs in the queues of* SRPT *and* $\mathcal{A}$ *at time* $t$. *The shaded jobs belong to* $\mathcal{A}$*'s queue* $\widetilde{Q}_t$*, while the unshaded jobs belong to* SRPT*'s queue* $Q_t$. *The arrows indicate that for any* $i > 1$ *the remaining processing time of the job of rank* $i$ *in* $Q_t$ *is at least that of the job of rank* $i-1$ *in* $\widetilde{Q}_t$.

to establish the following relationship between the queues of SRPT and $\mathcal{A}$: for any $i > 1$, the remaining processing time of the job of rank $i$ in $Q_t$ is *at least* the remaining processing time of the job of rank $i-1$ in $\widetilde{Q}_t$. The preceding claim is formally stated in Lemma 2.8 and illustrated in Figure 1.

We establish Lemma 2.8 by relating appropriately defined "prefixes" of the two queues $Q_t$ and $\widetilde{Q}_t$. For any time $t$ and positive integer $i$, let $P_t(i)$ denote the set of jobs in $Q_t$ with remaining processing time at most $i$. For any time $t$ and positive integer $i$, let $V_t(i)$ denote the sum of the remaining processing times of all jobs in $P_t(i)$. We call $V_t(i)$ the *volume* of jobs in $Q_t$ with remaining processing time at most $i$.

LEMMA 2.7. *For any time* $t$ *and any positive integer* $i$, $V_t(i)$ *is at most* $\widetilde{V}_t(i)+i$.

*Proof.* The proof is by induction on $t$. The induction basis holds trivially since for all $i$, $V_0(i) = \widetilde{V}_0(i)$. For the induction hypothesis, we assume that the claim is true at the end of step $t-1$. We now establish the induction step for time step $t$. Let $i$ be any positive integer. We first note that the release of new jobs at the start of step $t$ does not change $V_t(i) - \widetilde{V}_t(i)$ for any $i$. Now let $X$ denote the union of the set $P_{t-1}(i)$ and the set of jobs with processing time at most $i$ that are released at time $t$.

We consider two cases, depending on whether $X$ is empty. In the case where $X$ is nonempty, SRPT executes one job from $X$. Since $\mathcal{A}$ performs at most one unit of work, we can invoke the induction hypothesis and obtain that $V_t(i) \le \widetilde{V}_t(i) + i$. We now consider the case where $X$ is empty. In this case, at most one job in $Q_t$ can have remaining processing time at most $i$ at the end of step $t$; this happens when a job with remaining processing time $i+1$ gets processed and enters $P_t(i)$. Therefore, we have $V_t(i) \le i \le \widetilde{V}_t(i) + i$. This completes the proof of the desired claim. ∎

LEMMA 2.8. *For any time* $t$ *and any integer* $k > 1$, *we have* $\rho_t(q_{t,k}) \ge \widetilde{\rho}_t(\widetilde{q}_{t,k-1})$.

*Proof.* We first prove that for all $k > 1$, $\rho_t(q_{t,k}) \ge \widetilde{\rho}_t(\widetilde{q}_{t,k-1})$. The proof is by contradiction. Let, if possible, $k > 1$ be the smallest integer such that $\rho_t(q_{t,k}) < \widetilde{\rho}_t(\widetilde{q}_{t,k-1})$. Let $b$ denote $\rho_t(q_{t,k})$. It follows that $|P_t(b)| \ge k$, while $|\widetilde{P}_t(b)| = k-2$. We thus have the following inequality:

$$\begin{aligned} V_t(b) &\ge \sum_{1\le i\le k} \rho_t(q_{t,i}) \\ &= \rho_t(q_{t,1}) + \rho_t(q_{t,k}) + \sum_{2\le i<k} \rho_t(q_{t,i}) \end{aligned}$$

$$\begin{aligned}&\geq \rho_t(q_{t,1}) + b + \sum_{1\leq i<k-1} \widetilde{\rho}_t(\widetilde{q}_{t,i})\\ &= \rho_t(q_{t,1}) + b + \widetilde{V}_t(b)\\ &> b + \widetilde{V}_t(b),\end{aligned}$$

thus contradicting Lemma 2.7. (In the third step, we use the assumption that for $1 < i < k$, $\rho_t(q_{t,i}) \geq \widetilde{\rho}_t(\widetilde{q}_{t,i-1})$. In the final step, we use the fact that $\rho_t(q_{t,1})$ is positive.) □

Corollary 2.9. *For any time $t$, we have $\beta_t \leq \widetilde{\beta}_t + \frac{1}{\rho_t(q_{t,1})}$.*

*Proof.* The desired claim follows from Lemma 2.8:

$$\begin{aligned}\beta_t - \widetilde{\beta}_t &= \sum_{k\geq 1}\left(\frac{1}{\rho_t(q_{t,k})} - \frac{1}{\widetilde{\rho}_t(\widetilde{q}_{t,k})}\right)\\ &\leq \frac{1}{\rho_t(q_{t,1})} + \sum_{k\geq 2}\left(\frac{1}{\rho_t(q_{t,k})} - \frac{1}{\widetilde{\rho}_t(\widetilde{q}_{t,k-1})}\right)\\ &\leq \frac{1}{\rho_t(q_{t,1})},\end{aligned}$$

since $\rho_t(q_{t,k}) \geq \widetilde{\rho}_t(\widetilde{q}_{t,k-1})$ for all $k \geq 2$. □

We are now ready to prove Lemma 2.5.

*Proof of Lemma* 2.5. Consider an arbitrary time step $t$. Let $\mathcal{B}$ be a scheduling algorithm that minimizes the sum of the reciprocals of the processing times of the jobs in the queue at time $t$. Let $\alpha'_t$ denote the sum of the reciprocals of the processing times of the jobs in $\mathcal{B}$'s queue at time $t$.

We now argue that $\mathcal{B}$ can be chosen such that for every job $j$ in $\mathcal{B}$'s queue, the remaining processing time at time $t$ is the same as the processing time $p(j)$. If not, we can define a schedule $\mathcal{C}$ that mimics $\mathcal{B}$ except that $\mathcal{C}$ never processes any of the jobs that remain in $\mathcal{B}$'s queue at time $t$. Clearly, $\mathcal{C}$'s queue at time $t$ contains exactly the same jobs as $\mathcal{B}$'s. Moreover, none of the jobs in $\mathcal{C}$'s queue at time $t$ have been processed prior to time $t$. Since the remaining processing time of every job in $\mathcal{C}$'s queue is the same as the actual processing time, we obtain the desired inequality from Corollary 2.9:

$$\beta_t \leq \alpha'_t + \frac{1}{\rho_t(q_{t,1})} \leq \widetilde{\alpha}_t + \frac{1}{\rho_t(q_{t,1})}. \qquad \square$$

**2.2. Lower bound.** In this section, we prove Theorem 2.3. Consider an instance in which a job of size $\ell_1$ arrives in time step 0 and a job of size $\ell_2 \ll \ell_1$ arrives in time steps $\ell_1 + \ell_2(i-1) + 1$ for $0 \leq i < n-1$. An SRPT schedule will process the large job until time $\ell_1 - \ell_2 + 1$ when the first small job arrives. Since the remaining processing time of the large job $(\ell_2 - 1)$ is smaller than the size of the small job, SRPT continues processing the large job until it is finished. Thereafter, the $n-1$ small jobs are processed one after another. The total stretch of the SRPT schedule is $n + (n-1)(\ell_2-1)/\ell_2 = 2n - 1 - (n-1)/\ell_2$. Given $\varepsilon > 0$, we set $\ell_2 > 2(n-1)/\varepsilon$ so that the total stretch of SRPT is at least $2n - 1 - \varepsilon/2$.

An alternative schedule is to complete the small jobs immediately upon their arrival, and complete the large job at time $\ell_1 + (n-1)\ell_2$. This yields a total stretch of $n + (n-1)\ell_2/\ell_1$. We set $\ell_1 > 2(n-1)\ell_2/\varepsilon$ so that the preceding schedule has total stretch at most $n + \varepsilon/2$.

Thus, we have $S(\mathcal{J}) \geq S^*(\mathcal{J}) + n - 1 - \varepsilon$. This completes the proof of Theorem 2.3.

**3. Analysis of SRPT for multiprocessor scheduling.** This section analyzes the competitiveness of SRPT on an $m$-processor machine, where $m$ is any integer greater than 1. For any instance $\mathcal{J}$, let $S(\mathcal{J})$ and $S^*(\mathcal{J})$ denote the total stretch of SRPT and an optimal schedule, respectively. The main result of this section is the following upper bound on the total stretch achieved by SRPT, which is proved in section 3.1.

THEOREM 3.1. *For any instance $\mathcal{J}$ on multiprocessors, $S(\mathcal{J})$ is at most $3S^*(\mathcal{J})+(6+2\sqrt{6})|\mathcal{J}|$. Thus, the competitive ratio of* SRPT *for multiprocessors is at most $9+2\sqrt{6} \leq 14$.*

We can also show a constant-factor lower bound on the competitive ratio of SRPT for multiprocessors. The lower bound is proved in section 3.2.

THEOREM 3.2. *For any even integer $m > 1$ and any positive real $\varepsilon$, there exists a scheduling instance $\mathcal{J}$ for an $m$-processor machine such that $S(\mathcal{J})$ is at least $(2.5-\varepsilon)S^*(\mathcal{J})$.*

**3.1. Upper bound.** A crucial component of our upper bound proof for uniprocessors in section 2 is the property of SRPT's queue established in Lemma 2.8. Lemma 2.8 implies that, given any scheduling algorithm $\mathcal{A}$ and any time $t$, we can map each job $j$ in SRPT's queue except the one with the smallest remaining processing time to a unique job in $\mathcal{A}$'s queue at time $t$ that has remaining processing time at most that of $j$. For multiprocessors, however, a similar claim does not hold. In fact, even for a two-processor system, one can construct an instance for every positive integer $k$ such that the following claim holds: there exists a time step $t$ when the number of jobs in SRPT's queue at time $t$ is $k$ while the queue in the optimal schedule at time $t$ is empty. A nice example of such an instance is presented in [18], where it is used to derive a lower bound on the competitive ratio of SRPT with respect to total flow time.

While establishing an upper bound on the total flow time of SRPT, [18] provides a characterization of the queue of SRPT that implies an upper bound on the *number of jobs* in the queue of SRPT at each step. We note that the total flow time of a schedule is the sum, taken over every time step $t$, of the number of jobs in the queue at time $t$. The analysis of [18] does not suffice for our purposes, however, because the total stretch takes into account the sum of the *reciprocals of the processing times* of the jobs in the queue at any time step. In our analysis of SRPT that follows, we perform a more careful comparison between SRPT and an optimal schedule in terms of the relative distribution of the remaining processing times of the jobs in the two queues at each time step. We are able to establish a *partial* mapping from the jobs in SRPT's queue to the jobs in another schedule such that the following property of the mapping holds: the remaining processing time of a mapped job in SRPT's queue is at least that of the job to which it is mapped. This mapping, together with an appropriate characterization of the unmapped jobs in SRPT's queue, then enables us to place an upper bound on the exact or "amortized" contribution of the jobs in the queue at each time step.

As in the case of uniprocessor scheduling, we rank all of the jobs in $Q_t$ in nondecreasing order of their remaining processing times, breaking ties according to an arbitrary ordering of the job IDs whenever necessary. Let $F_t$ denote the set of jobs in $Q_t$ ranked at most $m$. Let $U_t$ denote the set $Q_t \setminus F_t$.

Analogous to the uniprocessor case, we split the sum $S(\mathcal{J})$ into two parts $S_1(\mathcal{J})$ and $S_2(\mathcal{J})$:

$$S_1(\mathcal{J}) = \sum_t \sum_{j \in F_t} \frac{1}{p(j)} \quad \text{and}$$

$$S_2(\mathcal{J}) = \sum_t \sum_{j \in U_t} \frac{1}{p(j)}.$$

By invoking the same argument used in proving Lemma 2.4, we show that $S_1(\mathcal{J})$ equals $|\mathcal{J}|$.

LEMMA 3.3. *For any instance $\mathcal{J}$, $S_1(\mathcal{J}) = |\mathcal{J}|$.*

*Proof.* Since each job in $F_t$ is processed at time $t$, a job $j$ is in $F_t$ for exactly $p(j)$ steps. The desired claim follows. ∎

We now consider the jobs in $U_t$. For any time $t$ and positive integer $i$, let $P_t(i)$ denote the set of jobs in $Q_t$ with remaining processing time at most $i$. We define the *volume* of any subset $X$ of $Q_t$ as the sum of the remaining processing time of the jobs in $X$ at time $t$. For any time $t$ and positive integer $i$, let $V_t(i)$ denote the volume of $P_t(i)$.

The following lemma is an easy generalization of Lemma 2.7. (As in section 2, we follow the notational convention that the "tilde" version of a variable defined for SRPT denotes the corresponding variable for $\mathcal{A}$.)

LEMMA 3.4. *For any time $t$ and any positive integer $i$, $V_t(i)$ is at most $\widetilde{V}_t(i)+mi$.*

*Proof.* The proof is by induction on $t$. The induction basis holds trivially since, for all $i$, $V_0(i) = \widetilde{V}_0(i)$. For the induction hypothesis, we assume that the claim is true at the end of step $t-1$. We now establish the induction step for time step $t$. Let $i$ be any positive integer. We first note that the release of new jobs at the start of a step does not change $V_t(i) - \widetilde{V}_t(i)$ for any $i$. Let $X$ denote the union of the set $P_{t-1}(i)$ and the set of jobs with processing time at most $i$ that are released at time $t$.

We consider two cases, depending on whether $|X|$ is at least $m$. In the case where $|X|$ is at least $m$, all of the jobs processed by SRPT are in $X$. Since $\mathcal{A}$ can perform at most $m$ units of work, we invoke the induction hypothesis and obtain that $V_t(i) \le \widetilde{V}_t(i) + mi$. We now consider the case where $|X|$ is less than $m$. In this case, the number of jobs not in $X$ that are processed in step $t$ is at most $m-|X|$. Therefore, at most $m-|X|$ jobs outside of $X$ may have remaining processing time at most $i$ after time step $t$. We thus obtain that $V_t(i) \le mi \le \widetilde{V}_t(i) + mi$. ∎

Given any schedule $\mathcal{A}$ and any time $t$, we define a partial mapping $f_t$ from $U_t$ to $\widetilde{Q}_t$ that satisfies properties (P1) and (P2) defined below. Let $\text{MPD}_t$ (resp., $\text{UMPD}_t$) denote the set of jobs in $U_t$ that are mapped (resp., unmapped) by $f_t$.

(P1) For each job $j$ in $\text{MPD}_t$, $\widetilde{\rho}_t(f_t(j)) \le \rho_t(j)$.

(P2) For any $i > 0$, the total volume of the jobs in $\text{UMPD}_t \cap P_t(i)$ is at most $mi$.

LEMMA 3.5 (partial mapping). *For any schedule $\mathcal{A}$ and any time $t$, there exists an injective mapping $f_t$ from $U_t$ to $\widetilde{Q}_t$ that satisfies properties* (P1) *and* (P2).

*Proof.* We describe a procedure for defining the mapping $f_t$. This procedure runs in phases, starting from phase 1. Let $X$ denote the set of jobs in $U_t \cap P_t(i)$ with remaining processing time equal to $i$. Let $Y$ denote the set of jobs in $\widetilde{P}_t(i)$ to which no job in $U_t$ has been mapped at the start of phase $i$. (At the start of phase 1, $Y$ is $\emptyset$.) In phase $i$, we map as many jobs as possible in $X$ to jobs in $Y$. The mapping is illustrated in Figure 2.

We now argue that for every $i$ the following two properties hold at the start of phase $i$: (i) for each job $j$ in $U_t$ that has been mapped, $\widetilde{\rho}_t(f_t(j)) \le \rho_t(j)$; (ii) for $0 < k < i$, the total volume of the unmapped jobs in $U_t \cap P_t(k)$ is at most $mk$.

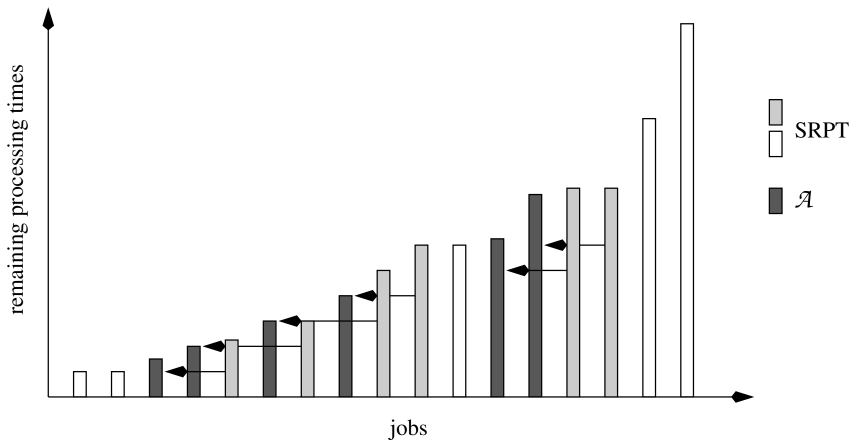


FIG. 2. *The remaining processing times of the jobs in the queues of* SRPT *and $\mathcal{A}$ at time $t$. The arrows depict the mapping $f_t$. The darkly shaded jobs belong to $\mathcal{A}$'s queue, while the lightly shaded and unshaded jobs belong to* SRPT*'s queue. The lightly shaded jobs are the mapped jobs in* SRPT*'s queue.*

Clearly, if the two properties hold at the end of the procedure when all of the jobs in $U_t$ have been considered, then the desired claim follows.

By the definition of the mapping, it is evident that property (i) holds at the start of every phase. We now prove property (ii). The proof is by induction on $i$. For the induction basis, $i = 1$ and property (ii) holds vacuously. We now assume that property (ii) holds at the start of phase $i$ and consider the effect of phase $i$.

By property (ii) of the induction hypothesis, it follows that for $0 < k < i$ the total volume of the unmapped jobs in $U_t \cap P_t(k)$ is at most $mk$. Thus, it suffices to prove that at the start of phase $i+1$, the total volume of the unmapped jobs in $U_t \cap P_t(i)$ is at most $mi$. We consider two cases. The first case is when every job in $\widetilde{P}_t(i)$ is the image of some job after phase $i$. By property (i), each job in $\text{MPD}_t$ is mapped to a job in $\widetilde{Q}_t$ with no larger remaining processing time. Therefore, it follows from Lemma 3.4 that the volume of unmapped jobs in $U_t \cap P_t(i)$ is at most $mi$. The second case is when there exists a job in $\widetilde{P}_t(i)$ to which no job is mapped at the end of phase $i$. In this case, we note that every job in $U_t$ with remaining processing time equal to $i$ is mapped in phase $i$. Therefore, we invoke property (ii) of the induction hypothesis to obtain that the total volume of the jobs in $U_t \cap P_t(i)$ that are unmapped at the end of phase $i$ is at most $m(i-1) \leq mi$. This completes the induction step and the proof of the lemma. □

Properties (P1) and (P2) help in placing upper bounds on the contribution of the mapped jobs and unmapped jobs, respectively, to $S_2(\mathcal{J})$. While Lemma 3.5 holds for any schedule $\mathcal{A}$, the particular mapping $f_t$ that we employ for the remainder of this proof is with respect to a schedule $\mathcal{B}$ that minimizes the sum of the reciprocals of the processing times of the jobs in the queue at time $t$. Let $\alpha'_t$ denote the sum of the reciprocals of the processing times of the jobs in $\mathcal{B}$'s queue at time $t$. As in Lemma 2.5, we can assume without loss of generality that none of the jobs that belong to $\mathcal{B}$'s queue at time $t$ have been processed until time $t$. Therefore, for each job $j$ in $\mathcal{B}$'s queue at time $t$, the remaining processing time equals $p(j)$. By property (P1) of Lemma 3.5, we know that $\rho_t(j) \geq p(f_t(j))$. Since $f$ is injective, we obtain the following lemma about the mapped jobs in $Q_t$.

LEMMA 3.6 (mapped jobs). *For any time $t$, we have $\sum_{j \in \mathrm{MPD}_t} \frac{1}{\rho_t(j)} \le \alpha'_t$.* □

We now consider the unmapped jobs. We begin by establishing a technical lemma that is helpful in placing an upper bound on the contribution of the unmapped jobs. (The reader may choose to skip the proof of this lemma on their first pass through this section.)

LEMMA 3.7. *Let $m$ be a positive integer. Let $X$ be a multiset of positive reals that satisfies the following condition.*

(C) *For any positive real $x$ the sum of the elements in $X$ with value at most $x$ is at most $mx$.*

*Then the following inequality holds true for any positive real $z$:*

$$\sum_{x \in X, x \ge z} \frac{1}{x} \le \frac{2m-1}{z}.$$

*Proof.* We first consider the case $m = 1$. By condition (C), this case implies that $X$ has at most one element. Therefore, the desired inequality follows trivially.

In the remainder of the proof, we assume that $m > 1$. Let $v(X)$ denote the term $\sum_{x \in X, x \ge z} \frac{1}{x}$. Consider the following greedy algorithm for constructing $X$ with the goal of maximizing $v(X)$. Initially set $X$ to $\emptyset$. Iteratively, add to $X$ the smallest element greater than or equal to $z$ that can be added without violating the condition on $X$ as stated in the lemma. Let $x_i$ be the $i$th element added to $X$. We claim that $x_i$ may be defined as follows:

$$x_i = \begin{cases} z & \text{if } 1 \le i \le m, \\ \frac{\sum_{1 \le j < i} x_j}{m-1} & \text{otherwise.} \end{cases}$$

We first prove the following two claims by induction on $i$: (i) $x_i \ge x_j$ for $1 \le j < i$, and (ii) the multiset $\{x_j : 1 \le j \le i\}$ is a valid solution for $X$. For the base case, we consider $1 \le i \le m$. Since $x_i = z$, the desired claim holds trivially for the base case. We now assume that the desired claim holds for $i-1$. Consider iteration $i$. The element $x_i$ equals $(\sum_{1 \le j < i} x_j)/(m-1)$, which is at least $(\sum_{1 \le j < i-1} x_j)/(m-1)$ and hence at least $x_{i-1}$, thus establishing part (i) of the induction step. For part (ii) of the induction step, we note that part (i) implies that we only need to check whether $\sum_{1 \le j \le i} x_j \le m x_i$. By the definition of $x_i$ it follows that $\sum_{1 \le j \le i} x_j = m x_i$, thus ensuring that the multiset $\{x_j : 1 \le j \le i\}$ is a valid solution for $X$. This completes the proofs of claims (i) and (ii).

We next show that $X^* = \{x_j : j \ge 1\}$ is an optimal choice for $X$. If possible, let $Y$ be a different optimal solution. Let $y_i$ denote the $i$th smallest element of $Y$, where we break ties arbitrarily. Let $i$ be the smallest integer such that $y_i \ne x_i$. We now derive a contradiction in each of the following two cases: $y_i > x_i$ and $y_i < x_i$. If $y_i > x_i$, we can replace the element $y_i$ in $Y$ by the element $x_i$ and obtain a valid solution yielding a value strictly larger than $v(Y)$, thus contradicting the assumption that $v(Y)$ is optimal. If $y_i < x_i$, then since $Y$ is a valid solution, so is the multiset $\{y_j : 1 \le j \le i\}$. This implies the following inequality:

$$\begin{aligned} \sum_{1 \le j \le i} y_j &= y_i + \sum_{1 \le j < i} x_j \\ &= y_i + (m-1)x_i \\ &> y_i + (m-1)y_i \\ &= m y_i, \end{aligned}$$

thus violating the condition stated in the lemma. (The argument for the second step is the following: since all of the elements in $Y$ are at least $z$, we have $i > m$, implying that $(m-1)x_i = \sum_{1 \le j < i} x_j$.)

To complete the proof of the lemma, we now compute $v(X^*)$. We first note that for $i > m$, $x_i$ is equal to $z(m/m-1)^{i-m}$. Therefore, we can calculate $v(X^*)$ as follows:

$$\begin{aligned} v(X^*) &= \frac{m}{z} + \sum_{i>m} \frac{1}{z}\left(\frac{m-1}{m}\right)^{i-m} \\ &= \frac{m}{z} + \sum_{j>0} \frac{1}{z}\left(\frac{m-1}{m}\right)^{j} \\ &= \frac{2m-1}{z}. \qquad \square \end{aligned}$$

In the following lemma, we invoke the inequality established above to place an upper bound on the sum of the reciprocals of the remaining processing times of certain subsets of unmapped jobs.

LEMMA 3.8. *For any positive integer $i$ and any step $t$, we have*

$$\sum_{j \in \mathrm{UMPD}_t, \rho_t(j) \ge i} \frac{1}{\rho_t(j)} \le \frac{2m-1}{i}.$$

*Proof.* We invoke Lemma 3.7, with $X$ equal to the set $\{j \in \mathrm{UMPD}_t : \rho_t(j) \ge i\}$, to prove the desired claim. We note that condition (C) of Lemma 3.7 holds due to property (P2) of Lemma 3.4. The claim of Lemma 3.7 is the desired inequality. $\square$

Lemma 3.8 indicates that if the rank of every unmapped job exceeds the rank of $\Omega(m)$ mapped jobs at any time step $t$, then the total stretch contribution of the unmapped jobs at $t$ can be upper bounded by a constant factor times the sum of the reciprocals of the remaining processing times of the mapped jobs, which in turn is bounded by Lemma 3.6. This suggests classifying the unmapped jobs at any time $t$ into two groups as follows. A job $j$ is *marked at time $t$* if $j \in \mathrm{UMPD}_t$ and the number of mapped jobs in $P_t(\rho_t(j))$ is at most $m$. Note that whether a job is marked depends on the particular time $t$; thus, for instance, a job may be marked at one time step and may be unmarked at a subsequent step. We call the first time when a job gets marked the *marking time* of the job; the marking time of a job that is never marked may be set to $\infty$. Let $\mathrm{MKD}_t$ and $\mathrm{UMKD}_t$ denote the sets of marked and unmarked jobs, respectively, at time $t$. We first account for the contribution of unmarked jobs by relating it to the contribution of mapped jobs of lesser rank.

LEMMA 3.9 (unmarked jobs). *For any time $t$, we have*

$$\sum_{j \in \mathrm{UMKD}_t} \frac{1}{p(j)} \le \sum_{j \in \mathrm{MPD}_t} \frac{2}{\rho_t(j)}.$$

*Proof.* For any mapped job $j$ in $Q_t$, let $X(j)$ denote the set of unmarked jobs in $Q_t$ that have rank greater than that of $j$. Since each job in $X(j)$ is unmapped, the following inequality follows from Lemma 3.8:

$$\sum_{j_1 \in X(j)} \frac{1}{\rho_t(j_1)} \le \frac{2m-1}{\rho_t(j)}. \tag{3.1}$$

By definition, for each unmarked job $j_1$, there exist at least $m$ jobs $j$ such that $j_1$ is in $X(j)$. Therefore, we obtain the following inequality:

$$\begin{aligned}\sum_{j_1 \in \mathrm{UMKD}_t} \frac{1}{p(j_1)} &\le \sum_{j_1 \in \mathrm{UMKD}_t} \frac{1}{\rho_t(j_1)} \\ &\le \frac{1}{m} \sum_{j \in \mathrm{MPD}_t} \sum_{j_1 \in X(j)} \frac{1}{\rho_t(j_1)} \\ &\le \frac{1}{m} \sum_{j \in \mathrm{MPD}_t} \frac{2m-1}{\rho_t(j)} \\ &\le \sum_{j \in \mathrm{MPD}_t} \frac{2}{\rho_t(j)}.\end{aligned}$$

(The third inequality is derived by adding (3.1) over all the mapped jobs.) □

We now consider the marked jobs. We call a marked job $j$ *dependent* at time $t$ if the number of jobs in $F_t$ with release time of at least the marking time of $j$ is greater than $\gamma m$, where $\gamma$ is a constant in $(0, 1)$ to be specified later. A marked job that is not dependent at time $t$ is said to be *independent*. Let $\mathrm{DEP}_t$ and $\mathrm{IND}_t$ denote the set of dependent and independent jobs, respectively, at time $t$. We account for the contributions of the jobs in $\mathrm{IND}_t$ and $\mathrm{DEP}_t$ in the following manner. In Lemma 3.10, we show that any job $j$ is independent during $O(p(j))$ steps only. This implies that the total contribution of the independent jobs during the execution of SRPT is $O(|\mathcal{J}|)$. Next, in Lemma 3.11, we show that the total contribution of the marked dependent jobs is within a constant factor of the contribution of the jobs in $F_t$, which we have already shown in Lemma 3.3 to be equal to $|\mathcal{J}|$.

LEMMA 3.10 (independent jobs). *Any job $j$ is independent during at most $2p(j)/(1-\gamma)$ time steps.*

*Proof.* Since any job $j$ is independent only when it is marked, we only need to consider time steps beginning from the marking time of $j$. Let $t$ denote the marking time of $j$. Let $X$ denote the set of jobs in $Q_t$ that have rank less than that of $j$. Since $j$ is marked at time $t$, it follows that the number of mapped jobs in $X$ is at most $m$. Moreover, by property (P2) of Lemma 3.5, the volume of all of the unmapped jobs in $X$ at time $t$ is at most $m\rho_t(j) \le mp(j)$. Therefore, the volume of all the jobs in $X$ is at most $2mp(j)$.

Consider any time step $t' \ge t$ when $j$ is independent. It follows from the definition of independence that there are at most $\gamma m$ jobs in $F_{t'}$ with release time at least $t$. Since $j$ is not in $F_{t'}$, it follows that there are at least $(1-\gamma)m$ jobs in $X$ that are in $F_{t'}$. Thus, the volume of the jobs in $X$ decreases by at least $(1-\gamma)m$ in time step $t_1$. Once all of the jobs in $X$ are completed, $j$ can be never be independent. Since the total volume of $X$ at time $t$ is at most $2mp(j)$, the number of time steps when $j$ is independent is at most $2p(j)/(1-\gamma)$. □

We next consider the dependent jobs.

LEMMA 3.11 (dependent jobs). *For any time $t$, we have*

$$\sum_{j \in \mathrm{DEP}_t} \frac{1}{p(j)} \le \frac{3}{\gamma} \sum_{j \in F_t} \frac{1}{p(j)}.$$

*Proof.* Consider a job $j$ in $F_t$. Let $X(j)$ be the set of jobs in $\mathrm{DEP}_t$ whose marking time is at most the release time of $j$. Since every job in $X(j)$ has rank greater than

$m$ and hence greater than that of $j$, it follows that at the time of the release of $j$ the remaining processing time of every job in $X(j)$ is at least $p(j)$; that is, for any $j_1$ in $X(j)$, we have $\rho_{r(j)}(j_1) \geq p(j)$. Moreover, if $Y(j)$ denotes the set of jobs in $X(j)$ that have been processed at any time during the time interval $[r(j), t]$, then $|Y(j)|$ is at most $m - 1$ because every job in $X(j)$ has rank greater than that of $j$.

Consider the jobs in $X(j) \setminus Y(j)$. For every job $j_1$ in $X(j) \setminus Y(j)$, we have $\rho_t(j_1) = \rho_{r(j)}(j_1) \geq p(j)$. Moreover, since every job in $X(j)$ is unmapped, we obtain the following inequality from Lemma 3.8:

$$\sum_{j_1 \in X(j) \setminus Y(j)} \frac{1}{p(j_1)} \leq \frac{2m}{p(j)}. \tag{3.2}$$

We now consider the jobs in $Y(j)$. Since there are at most $m - 1$ jobs in $Y(j)$, each with processing time at least $p(j)$, we have the following inequality:

$$\sum_{j_1 \in Y(j)} \frac{1}{p(j_1)} \leq \frac{m-1}{p(j)}. \tag{3.3}$$

Equations 3.2 and 3.3 together yield the following inequality for each job $j$ in $F_t$:

$$\sum_{j_1 \in X(j)} \frac{1}{p(j_1)} \leq \frac{3m}{p(j)}. \tag{3.4}$$

Now consider a job $j_1$ in $\text{DEP}_t$. Since $j_1$ is dependent, there are more than $\gamma m$ jobs $j$ in $F_t$ such that $j_1$ is in $X(j)$. Thus, if we add up both the left-hand and right-hand sides of (3.4) over all jobs in $F_t$, $j_1$ appears at least $\lceil \gamma m \rceil$ times on the left-hand side. Therefore, we obtain the following inequality, that establishes the desired claim:

$$\sum_{j_1 \in \text{DEP}_t} \frac{1}{p(j_1)} \leq \frac{3}{\gamma} \sum_{j \in F_t} \frac{1}{p(j)}. \qquad \square$$

Putting Lemmas 3.6, 3.9, 3.10, and 3.11 together, we obtain the following main lemma.

LEMMA 3.12. *For any instance $\mathcal{J}$, we have*

$$\sum_t \sum_{j \in U_t} \frac{1}{p(j)} \leq 3S^*(\mathcal{J}) + \left(\frac{2}{1-\gamma} + \frac{3}{\gamma}\right) |\mathcal{J}|.$$

*Proof.* We establish the desired inequality as follows:

$$\begin{aligned}
\sum_t \sum_{j \in U_t} \frac{1}{p(j)} &= \sum_t \left( \sum_{j \in \text{MPD}_t} \frac{1}{p(j)} + \sum_{j \in \text{IND}_t} \frac{1}{p(j)} + \sum_{j \in \text{DEP}_t} \frac{1}{p(j)} + \sum_{j \in \text{UMKD}_t} \frac{1}{p(j)} \right) \\
&\leq \sum_t \sum_{j \in \text{MPD}_t} \frac{3}{\rho_t(j)} + \sum_j \sum_{t: j \in \text{IND}_t} \frac{1}{p(j)} + \frac{3}{\gamma} \sum_t \sum_{j \in F_t} \frac{1}{p(j)} \\
&\leq \sum_t 3\alpha'_t + \frac{2|\mathcal{J}|}{1-\gamma} + \frac{3}{\gamma} \sum_t \sum_{j \in F_t} \frac{1}{p(j)} \\
&\leq 3S^*(\mathcal{J}) + \left(\frac{2}{1-\gamma} + \frac{3}{\gamma}\right) |\mathcal{J}|.
\end{aligned}$$

(In the second step, we invoke Lemmas 3.9 and 3.11. In the third step, we invoke Lemmas 3.6 and 3.10. The final step follows from Lemma 3.3.) ∎

The term $2/(1-\gamma)+3/\gamma$ attains a minimum value of $5+2\sqrt{6} \leq 10$ when $\gamma = 3-\sqrt{6}$. The main theorem now directly follows from Lemmas 3.3 and 3.12.

**3.2. Lower bound.** We now prove Theorem 3.2. We consider the case of two processors only; the argument can be easily extended to the case of $2m$ processors for any positive integer $m$. The main idea in the argument is similar to that used in [18]. Consider an instance that starts with three jobs, one of size $2^n$ and two of size $2^{n-1}$, all of which arrive at time 0. At time $2^n$, one job of size $2^{n-1}$ and two jobs of size $2^{n-2}$ are released. In general, for $1 \leq i \leq n$, at time $\sum_{i<j\leq n} 2^j$, one job of size $2^i$ and two jobs of size $2^{i-1}$ are released. Finally, at each of the $p$ steps of the interval $[2^{n+1}-2, 2^{n+1}+p-3]$, a pair of unit-size jobs are released. (Here $p$ is an integer that is specified below.) The optimal total stretch equals $4n+2p$. The total stretch achieved by SRPT is at least $3n+4p+p(1/2^{n-2}+1/2^{n-1}+\cdots+1/2)$, which is $3n+(5-1/2^{n-2})p$. For any constant $\varepsilon > 0$, we can choose $p$ sufficiently larger than $n$ so as to make the competitive ratio at least $2.5-\varepsilon$.

## 4. Lower bounds.

This section contains lower bounds on the competitive ratio of arbitrary online algorithms for uniprocessor and multiprocessor scheduling. Section 4.1 concerns uniprocessor scheduling, while section 4.2 concerns multiprocessor scheduling.

**4.1. Uniprocessor scheduling.** Our main result for lower bounds on online uniprocessor scheduling is the following.

THEOREM 4.1. *If there are only two distinct job sizes, there exists an optimal online algorithm for minimizing average stretch. If there are three or more distinct job sizes, the competitive ratio of any online algorithm is at least* 1.04.

*Proof.* Consider the case when there are only two distinct job sizes. We first observe that there is an optimal schedule in which all large (resp., small) jobs are processed in first-in-first-out (FIFO) order. Thus, at any time $t$, there is at most one partially processed job in each category, and that is the job at the head of the FIFO order in that category at that time; let $\ell$ and $s$ denote the job at the head of the FIFO order in the large and small categories, respectively. We need to decide which one of these two jobs must be processed. Suppose we complete the large job $\ell$ first, and follow it by processing the small job $s$; then the total stretch contributed by the two jobs is

$$\frac{t+\rho_t(\ell)-r(\ell)}{p(\ell)} + \frac{t+\rho_t(\ell)+\rho_t(s)-r(s)}{p(s)}.$$

On the other hand, if we complete the small job first, followed by the the large job, then the stretch contributed by the two jobs is

$$\frac{t+\rho_t(s)-r(s)}{p(s)} + \frac{t+\rho_t(\ell)+\rho_t(s)-r(\ell)}{p(\ell)}.$$

The former is desirable when $\rho_t(\ell)/p(s) \leq \rho_t(s)/p(\ell)$, that is, when $\rho_t(\ell)p(\ell) \leq \rho_t(s)p(s)$.

We now argue that the following online algorithm is optimal: at each time step $t$, process a job $i$ with the smallest $\rho_t(i)p(i)$. The proof is by contradiction. Suppose that there exists an optimal schedule in which there is a time step $t$ at which a job $i$

with the smallest $\rho_t(i)p(i)$ is not processed. Let $t$ be the earliest such time step. Let $s$ and $\ell$ denote the jobs at the head of the FIFO order in the small and large categories, respectively, at time $t$. Let us consider the case that $\rho_t(\ell)p(\ell) < \rho_t(s)p(s)$, yet job $s$ is processed at time $t$. Let $t'$ denote the completion time of job $\ell$ in the schedule. It then follows that the set of jobs that complete during the interval $[t, t')$ are all small jobs (this set is nonempty since $s$ belongs to it). Let $j$ denote the last small job to complete in the interval $[t, t')$. Consider the schedule in which we swap the order of jobs $j$ and $\ell$ and complete $\ell$ before $j$; thus, the completion time of $\ell$ is decreased by at least $\rho_t(s)$, and the completion time of $j$ is increased by at most $\rho_t(\ell)$. As a result, the decrease in total stretch is at least

$$\frac{\rho_t(s)}{p(\ell)} - \frac{\rho_t(\ell)}{p(s)} > 0,$$

which contradicts the assumption that the given schedule is optimal. The analysis for the other case is symmetric, thus completing the proof of the first claim of the theorem.

We next consider the lower bound when there are three or more distinct job sizes. We give the proof of a weaker lower bound first. Consider three jobs $j_1$, $j_2$, and $j_3$, with release times 0, 4, and 5, respectively, and processing times 5, 2, and 1, respectively. Let instance $\mathcal{J}_1$ be the set $\{j_1, j_2\}$ and instance $\mathcal{J}_2$ be the set $\{j_1, j_2, j_3\}$. For $\mathcal{J}_1$, the optimal solution is to process $j_1$ for 4 steps, then process $j_2$ for 2 steps and then complete $j_1$. The optimal total stretch is $1 + 7/5 = 2.4$. For $\mathcal{J}_2$, the optimal solution is to process $j_1$ for 5 steps and thus complete it, then process $j_3$ for 1 step, and finally, process $j_2$. The optimal stretch value for this instance is 4. For instance $\mathcal{J}_1$, if the given online algorithm completes $j_1$ before $j_2$, then its stretch is at least 2.5, implying a competitive ratio of at least 1.04. If, on the other hand, the algorithm preempts $j_1$ at time 4 (i.e., at the start of the fifth step), then the best scenario for the algorithm with regard to instance $\mathcal{J}_2$ is to finish the jobs in one of the following orders: (i) $j_2$, $j_3$, $j_1$ or (ii) $j_3$, $j_2$, $j_1$. The stretch for case (i) is 4.6 and for case (ii) is 4.1. Therefore, the competitive ratio is at least 1.025.

We can generalize the above example as follows. Let $\ell$, $m$, and $s$ be the jobs with processing times $p(\ell)$, $p(m)$, and $p(s)$, respectively, where $p(\ell) > p(m) > p(s)$. Consider the instance in which $\ell$ arrives at time 0 and $m$ arrives at the end of time step $p(\ell) - k$, where $k \leq p(m)$. If the algorithm does not preempt the first job when the second job arrives, the competitive ratio is at least $(2 + (k/p(m)))/(2 + (p(m)/p(\ell)))$. Otherwise, job $s$ is presented at the end of time step $p(\ell)$. Then we can argue as above that any such algorithm has average stretch at least $\min\{3 + p(s)/p(m) + (p(m) + p(s))/p(\ell), 3 + (p(m) + p(s))/p(\ell) + (p(m) - k)/p(s)\}$, while the optimum average stretch is at most $3 + (k + p(s))/p_m$. The lower bound on the competitive ratio, thus obtained, is nearly maximized by setting $p(s) = 54$, $p(m) = 100$, $p(\ell) = 200$, and $k = 60$; this setting of parameters yields that the average stretch is at least 1.04. $\square$

**4.2. Multiprocessor scheduling.** For multiprocessors, we can establish a slightly stronger lower bound on the competitive ratio of any online algorithm.

THEOREM 4.2. *For any positive integer $m$, the competitive ratio of any online algorithm for any $2m$-processor machine is at least $7/6$.*

*Proof.* We first consider the case of a 2-processor machine. We consider two instances of the scheduling problem. One instance consists of three jobs, two of size 1 and one of size $\ell > 1$, all of which are released at time 0. The parameter $\ell$ is an integer, whose value is specified below. The optimal schedule for this instance is to schedule the two size-1 jobs in parallel and then schedule the size-$\ell$ job, thus yielding a total stretch of $3 + 1/\ell$. If the given online algorithm schedules the two unit-size jobs in sequential order on one of the processors, then it achieves a total stretch of 4, thus yielding a competitive ratio of $4\ell/(3\ell + 1)$.

If the online algorithm instead schedules the above instance optimally, then we consider its performance on another scheduling instance which consists of the same set of three jobs as above, together with a sequence of unit-size jobs, arriving in pairs at times $\ell$, $\ell+1$, ..., $\ell+n-1$. Since the incomplete size-$\ell$ job has at least one unit of processing time remaining, the online algorithm will be forced to delay the size-$\ell$ job until all of the size-1 jobs complete, thus leading to a total stretch of $2n+3+(n+1)/\ell$. (Note that we are making use of our assumption that time is discrete; thus, job arrivals and preemptions occur at integer time steps.) An alternative schedule for this instance is to schedule the first two unit-size jobs in sequence on one of the processors and the size-$\ell$ job on the other processor so that the sequence of pairs of unit-size jobs that are released from time $\ell$ onwards can be scheduled immediately upon arrival without delaying the size-$\ell$ job. The total stretch of this schedule is $2n + 4$. Therefore, the competitive ratio of the given online algorithm is at least $(2n\ell+3\ell+n+1)/(2n\ell+4\ell)$, which tends to $(2\ell + 1)/(2\ell)$ as $n$ tends to infinity.

We now choose integer $\ell$ so as to maximize the minimum of $4\ell/(3\ell + 1)$ and $(2\ell + 1)/(2\ell)$. A maximum of 7/6 is achieved at $\ell = 3$.

We next generalize the above proof to $2m$-processor machines, for $m > 1$. Again, we consider two instances of the scheduling problem. One instance consists of $3m$ jobs, $2m$ jobs of size 1 and $m$ jobs of size $\ell > 1$, all of which are released at time 0. The parameter $\ell$ is an integer, whose value is specified below. The optimal schedule for this instance is to schedule the $2m$ unit-size jobs in parallel on each of the $2m$ processors and then schedule the size-$\ell$ jobs on any $m$ of the processors, thus yielding a total stretch of $3m + m/\ell$. Let $k$ denote the number of size-$\ell$ jobs that the given online algorithm completes by time $\ell$. It follows that at least $k$ of the $2m$ unit-size jobs are delayed by a time unit and incur a stretch of 2. Furthermore, at least $m - k$ size-$\ell$ jobs are delayed by unit time. Thus the total stretch of the online algorithm is at least $3m + k + (m - k)/\ell$.

The second instance for which we evaluate the given online algorithm consists of the same set of $3m$ jobs as in the first instance, together with a sequence of $2nm$ unit-size jobs, arriving in groups of $2m$ jobs at each of the steps $\ell$, $\ell$+1, ..., $\ell$+$n$−1. For this instance, the total stretch objective forces the given online algorithm to schedule all of the $2nm$ jobs ahead of the $m-k$ size-$\ell$ incomplete jobs at time $\ell$. The total stretch of the online algorithm for the second instance is at least $2nm+3m+k+n(m-k)/\ell$. On the other hand, an alternative (optimal) schedule completes the first set of $3m$ jobs (that arrive at time 0) within time $\ell$ and then schedules the remaining $2nm$ unit-size jobs as they arrive. This schedule has a total stretch of $2nm + 4m$. As $n$ tends to infinity, the ratio $(2nm+3m+k+n(m-k)/\ell)/(2nm+4m)$ tends to $1+(m-k)/(2m\ell)$.

The competitive ratio of the online algorithm is thus at least

$$\begin{aligned}
&\min_k \max\left\{\frac{3m + k + (m-k)/\ell}{3m + m/\ell}, 1 + \frac{m-k}{2m\ell}\right\}\\
&\quad = \min_k \max\left\{1 + \frac{k - k/\ell}{3m + m/\ell}, 1 + \frac{m-k}{2m\ell}\right\}\\
&\quad = 1 + \min_k \max\left\{\frac{k - k/\ell}{3m + m/\ell}, \frac{m-k}{2m\ell}\right\}
\end{aligned}$$

$$\geq 1 + \frac{1}{2} \min_k \left[ \frac{1}{2\ell} + \frac{k}{m} \left( \frac{\ell - 1}{3\ell + 1} - \frac{1}{2\ell} \right) \right]$$
$$\geq 1 + \frac{\ell - 1}{6\ell + 2}.$$

(The penultimate step holds since the maximum of two numbers is at least the average of the two. The last step uses the inequality $(\ell - 1)/(3\ell + 1) \leq 1/2\ell$.) The term $1 + (\ell - 1)/(6\ell + 2)$ tends to 7/6 as $\ell$ tends to infinity. $\square$

**5. Discussion.** We have studied the online complexity of minimizing average stretch. Our results show that SRPT is attractive for both uniprocessor and multiprocessor cases since it simultaneously achieves optimal performance (up to constant factors) for average response time as well as average stretch. In general, any algorithm aimed at optimizing average response time or average stretch, and SRPT in particular, may starve jobs on worst case inputs. However, for certain applications such as serving web traffic, it appears that the adverse effect of starvations in SRPT is limited [4, 14]. In multiprocessors, a potential drawback of SRPT is the overhead it incurs due to migrations. Recall that SRPT may preempt a job at one processor and later resume the job at a different processor. A natural variant of SRPT that does not perform any migrations has been proposed in [2]. In recent work [5], it has been shown that the "migrationless" scheduling algorithm of [2] also achieves a constant-factor competitive ratio with respect to average stretch.

Many interesting problems remain open. In particular, the lower bounds may quite possibly be strengthened. Also, it will be of interest to analyze the algorithm that schedules the job $i$ with the smallest $p(i)\rho_t(i)$ at time $t$; for the special case of two job sizes, this algorithm is optimal, as we showed (see section 4). The average stretch metric may also be studied in other models, such as when preemption is not allowed.

In this paper, we have focused on online scheduling. The complexity of optimizing average stretch offline is open for both uniprocessors and multiprocessors. For uniprocessors, a PTAS is achievable, as shown recently in [7, 9]. For multiprocessors, the online constant-factor upper bounds that we have shown yield the best known approximations offline. Whether the offline problem is NP-hard is open, even for the multiprocessor case. The reduction used in the NP-completeness proof for minimizing average response time on 2-processors [10] may be helpful in this regard.

## REFERENCES

[1] F. Afrati, E. Bampis, C. Chekuri, D. Karger, C. Kenyon, I. Milis, M. Queyranne, M. Skutella, C. Stein, and M. Sviridenko, *Approximation schemes for scheduling to minimize average completion time with release dates*, in Proceedings of the 40th Annual IEEE Symposium on Foundations of Computer Science, New York, 1999, IEEE, Piscataway, NJ, pp. 32–43.

[2] B. Awerbuch, Y. Azar, S. Leonardi, and O. Regev, *Minimizing flow time without migration*, in Proceedings of the 31st Annual ACM Symposium on Theory of Computing, Atlanta, GA, 1999, ACM, New York, pp. 198–205.

[3] K. R. Baker, *Introduction to Sequencing and Scheduling*, Wiley, New York, 1974.

[4] N. Bansal and M. Harchol-Balter, *Analysis of SRPT scheduling: Investigating unfairness*, in Proceedings of ACM SIGMETRICS 2001/Performance 2001: Joint International Conference on Measurement & Modeling of Computer Systems, Cambridge, MA, 2001, pp. 279–290.

[5] L. Becchetti, S. Leonardi, and S. Muthukrishnan, *Scheduling to minimize average stretch without migration*, in Proceedings of the 11th Annual ACM-SIAM Symposium on Discrete Algorithms, San Francisco, 2000, SIAM, Philadelphia, pp. 548–557.

[6] M. A. Bender, S. Chakrabarti, and S. Muthukrishnan, *Flow and stretch metrics for scheduling continuous job streams*, in Proceedings of the 9th Annual ACM-SIAM Symposium on Discrete Algorithms, San Francisco, 1998, SIAM, Philadelphia, pp. 270–279.

[7] M. A. Bender, S. Muthukrishnan, and R. Rajaraman, *Improved algorithms for stretch scheduling*, in Proceedings of the 13th Annual ACM-SIAM Symposium on Discrete Algorithms, San Francisco, 2002, SIAM, Philadelphia, pp. 762–771.

[8] C. Chekuri and S. Khanna, *Approximation schemes for preemptive weighted flow time*, in Proceedings of the 34th Annual ACM Symposium on Theory of Computing, Montreal, 2002, ACM, New York, pp. 297–305.

[9] C. Chekuri, S. Khanna, and A. Zhu, *Algorithms for minimizing weighted flow time*, In Proceedings of the 33rd Annual ACM Symposium on Theory of Computing, Heraklion, Crete, 2001, ACM, New York, pp. 84–93.

[10] J. Du, J. Y.-T. Leung, and G. H. Young, *Minimizing mean flow time with release time constraint*, Theoret. Comput. Sci., 75 (1990), pp. 347–355.

[11] A. Goel, M. Henzinger, S. Plotkin, and E. Tardos, *Scheduling data transfers in a network and the set scheduling problem*, in Proceedings of the 31st Annual ACM Symposium on Theory of Computing, Atlanta, GA, 1999, ACM, New York, pp. 189–197.

[12] L. Hall, A. Schulz, D. Shmoys, and J. Wein, *Scheduling to minimize average completion time: Off-line and on-line algorithms*, Math. Oper. Res., 22 (1997), pp. 513–544.

[13] M. Harchol-Balter, M. Crovella, and C. Murta, *On choosing a task assignment policty for a distributed server system*, J. Parallel Distrib. Comput., 59 (1999), pp. 204–228.

[14] M. Harchol-Balter, M. Crovella, and S. Park, *The Case for SRPT Scheduling in Web Servers*, Technical Report MIT–LCS–TR–767, Laboratory for Computer Science, Massachusetts Institute of Technology, Cambridge, MA, 1998.

[15] R. Jain, *The Art of Computer Systems Performance Analysis*, John Wiley, New York, 1991.

[16] D. Karger, C. Stein, and J. Wein, *Scheduling algorithms*, in Handbook of Algorithms and Theory of Computation, M. Atallah, ed., CRC Press, Boca Raton, FL, 1999, Chapter 35.

[17] E. Lawler, J. Lenstra, A. Rinnooy Kan, and D. Shmoys, *Sequencing and scheduling: Algorithms and complexity*, in Handbooks in Operations Research and Management Science, Vol. 4: Logistics of Production and Inventory, S. Graves, A. Rinnooy Kan, and P. Zipkin, eds., North–Holland, Amsterdam, 1993, pp. 445–522.

[18] S. Leonardi and D. Raz, *Approximating total flow time on parallel machines*, in Proceedings of the 29th Annual ACM Symposium on Theory of Computing, El Paso, TX, 1997, ACM, New York, pp. 110–119.

[19] M. Mehta and D. J. DeWitt, *Dynamic memory allocation for multiple-query workloads*, in Proceedings of the 19th International Conference on Very Large Data Bases, Dublin, Ireland, 1993, pp. 354–367.

[20] A. Schulz and M. Skutella, *Scheduling-LPs bear probabilities: Randomized approximations for min-sum criteria*, in Proceedings of the 5th Annual European Symposium, Graz, Austria, R. Burkard and G. Woeginger, eds., Lecture Notes in Comput. Sci. 1284, Springer, New York, 1997, pp. 416–429.

[21] J. Sgall, *On-line scheduling*, in Online Algorithms, The State of the Art, A. Fiat and G. Woeginger, eds., Lecture Notes in Comput. Sci. 1442, Springer, New York, 1998, pp. 196–221.

[22] D. D. Sleator and R. E. Tarjan, *Amortized efficiency of list update and paging rules*, Comm. ACM, 28 (1985), pp. 202–208.

[23] H. Zhu, B. Smith, and T. Yang, *A scheduling framework for Web server clusters with intensive dynamic content processing*, Technical Report TRCS98-29, Department of Computer Science, University of California, Santa Barbara, CA, 1998.

SIAM J. COMPUT.
Vol. 34, No. 2, pp. 453–474

# TRAVELING WITH A PEZ DISPENSER (OR, ROUTING ISSUES IN MPLS)[*]

ANUPAM GUPTA[†], AMIT KUMAR[‡], AND RAJEEV RASTOGI[§]

**Abstract.** A new packet routing model proposed by the Internet Engineering Task Force is *MultiProtocol Label Switching*, or MPLS [B. Davie and Y. Rekhter, *MPLS: Technology and Applications*, Morgan Kaufmann (Elsevier), New York, 2000]. Instead of each router's parsing the packet network layer header and doing its lookups based on that analysis (as in much of conventional packet routing), MPLS ensures that the analysis of the header is performed just once. The packet is then assigned a *stack of labels*, where the labels are usually much smaller than the packet headers themselves. When a router receives a packet, it examines the label at the *top* of the label stack and makes the decision of where the packet is forwarded based solely on that label. It can pop the top label off the stack if it so desires, and can also push some new labels onto the stack, before forwarding the packet. This scheme has several advantages over conventional routing protocols, the two primary ones being (a) reduced amount of header analysis at intermediate routers, which allows for faster switching times, and (b) better traffic engineering capabilities and hence easier handling of quality of service issues. However, essentially nothing is known at a theoretical level about the performance one can achieve with this protocol, or about the intrinsic trade-offs in its use of resources.

This paper initiates a theoretical study of MPLS protocols, and routing algorithms and lower bounds are given for a variety of situations. We first study the routing problem on the line, a case which is already nontrivial, and give routing protocols whose trade-offs are close to optimality. We then extend our results for paths to trees, and thence onto more general graphs. These routing algorithms on general graphs are obtained by finding a *tree cover* of a graph, i.e., a small family of subtrees of the graph such that, for each pair of vertices, one of the trees in the family contains an (almost-)shortest path between them. Our results show tree covers of logarithmic size for planar graphs and graphs with bounded separators, which may be of independent interest.

**Key words.** network routing, tree covers, graph separators, MPLS routing, distance labeling, analysis of algorithms

**AMS subject classifications.** 68W40, 05C85, 05C78, 05C62, 05C90

**DOI.** 10.1137/S0097539702409927

## 1. Introduction.

In most conventional network routing protocols, a packet makes its way from source to destination in essentially the following way: when a packet reaches the router, the router analyzes the packet's header (which contains the destination address) and uses the results of this analysis to decide the next hop for the packet. These routing decisions are made locally and independently of other routers, based solely on the identity of the incoming edge and the analysis of the packet header. For example, routers using conventional IP forwarding typically look for a longest-prefix match between the destination address and the entries in the

[*]Received by the editors June 20, 2002; accepted for publication (in revised form) August 16, 2004; published electronically February 3, 2005. A preliminary abstract of this work appeared in the Proceedings of the 42nd Annual IEEE Symposium on Foundations of Computer Science [17]. Pez is a registered trademark of Pez Candy, Inc.

http://www.siam.org/journals/sicomp/34-2/40992.html

[†]Department of Computer Science, Carnegie Mellon University, Pittsburgh, PA 15213 (anupamg@cs.cmu.edu). Most of this research was done while the author was at Cornell University and Lucent Bell Labs.

[‡]Department of Computer Science, Indian Institute of Technology, Hauz Khas, New Delhi, India 110016 (amitk@cse.iitd.ernet.in). Most of this research was done while the author was at Cornell University, and was supported in part by Lucent Bell Labs and the ONR Young Investigator Award of Jon Kleinberg.

[§]Lucent Bell Labs, 600 Mountain Avenue, Murray Hill, NJ 07974-0636 (rastogi@research.bell-labs.com).

routing table to decide the next hop for the packet. In general, *each* router has to extract the information relevant to it from the (much longer) packet header. Furthermore, conventional routers are not designed to use information about the source of the packets from these headers.

An alternative to this routing model was proposed by the Internet Engineering Task Force (IETF) and is called *MultiProtocol Label Switching* (MPLS) [11, 20]. In MPLS, the analysis of the packet's network layer header is performed just once; this is done by the first router that receives the packet, called the *ingress router*. This analysis causes the packet to be assigned a *stack* of *labels*, where these labels are usually much smaller than the packet headers themselves [31, 30]. Each subsequent router now examines only the label at the *top* of the label stack, and makes its routing decision (i.e., the next hop) based solely on that label. It can then pop this label off the stack, if it so desires, and push some labels (maybe none) onto the stack, before sending the packet on its merry way. Note that, apart from looking at the top label, none of the subsequent routers perform any further analysis of the network layer header.

There are a number of advantages of MPLS over conventional network layer forwarding protocols. The first one, already alluded to above, is the elimination of header analysis at each hop. This allows the replacement of routers by simpler and faster switches, which merely perform the basic operations of label lookup and replacement. Furthermore, since the packet headers are analyzed by the ingress router when packets enter the system, the ingress router can possibly use additional information about packets to route different packets along different paths. This enables the routing of packets according to their desired quality of service. For example, packets corresponding to time-sensitive applications may be sent along different channels from regular data, which may be faster but more expensive. The ingress router may also use information about the source of the data, in addition to the destination address; this is something that is not possible in conventional routing protocols. In addition to these factors, traffic engineering and network control are important reasons for using MPLS rather than conventional routing schemes [3, 24, 12]. This is for two reasons: first, the time savings achieved by one-time analyses of packet headers allows us to perform fine-grained routing of packets; second, this fine-grained routing allows the entire route for the packet to be encoded very naturally on the stack. These features have made MPLS very popular among network providers and router designers, and companies like Cisco, Juniper, Lucent, and Nortel have been producing routers which support MPLS protocols [9, 25].

However, despite this popularity of MPLS, and the fact that it is becoming more widespread on the Internet, essentially nothing is known at a theoretical level about the performance achievable by MPLS, or about the intrinsic trade-offs in its use of resources. The basic question that this paper addresses is the following: what is the depth of the stack that is sufficient for routing in an $n$-node network, and how does this stack depth interact with the label size?

Note that a small number of labels is desirable, since the bandwidth reservation in networks is often done by creating a (virtual) channel for each label; a small number of labels thus ensures that traffic is not split too much, which usually implies a better bandwidth utilization. Furthermore, having a small set of labels causes the forwarding tables to be small, and makes the forwarding procedures simpler and faster. On the other hand, the size of the label stack translates directly to the size of the packet header, and hence small stacks are desirable as well. These goals naturally oppose each other, and the trade-offs between these resources are nontrivial. If the label size is $L$ and the stack depth is $s$, a simple counting argument indicates that $L^s \geq n$, the size of the network; it is not clear whether MPLS routing protocols can get close to this information theoretic bound.

Previous papers on distributed routing do not address such questions. Indeed, the idea of using a stack of labels is novel to MPLS, as is the all-important restriction that the routers can look only at the top of the stack when deciding the next hop for a packet. As a simple example, consider the question of routing on a path with 2 labels; with each intermediate router getting only one bit of information from the stack (instead of the entire $\log n$ bit destination address), is it possible to achieve a stack depth of $O(\log n)$? While we show in section 2 that this is indeed possible, the solution is not trivial and illustrates the issues that arise from this restricted model of access to the network header.

This paper initiates a theoretical study of the MPLS protocol. We give routing algorithms and lower bounds in a variety of situations. We first study the routing problem on a path, and give near-optimal protocols for this basic case. When combined with a suitable decomposition of trees into paths, these protocols then yield routing algorithms for trees. These results are then extended to more general classes of graphs by the use of *tree covers*. A tree cover is a set of subtrees of the graph such that for each pair of vertices one of the trees in the cover contains an (almost-)shortest path between the two vertices.

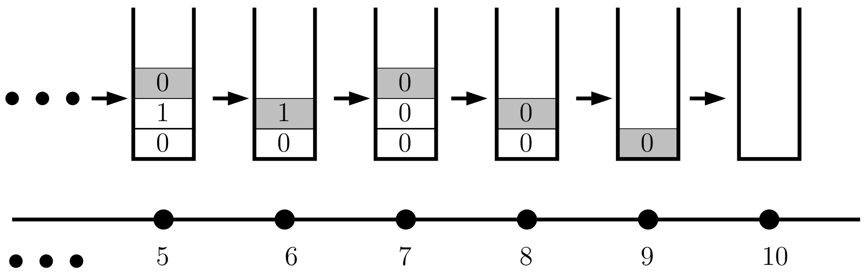
**1.1. The model.** The header of each *packet* consists of a *stack $S$* of *labels*. The labels are drawn from a set $\Sigma$ of size $L$, which is usually identified with the set $\{1, 2, \ldots, L\}$. The two quantities of interest are (a) the number of labels $L$ and (b) the maximum stack depths required for routing between any two vertices.

The network is represented by an undirected graph $G = (V, E)$, each node representing a *router* and running some routing protocol. A *routing protocol $\mathcal{A}$* has two components: a suite of *stack-making functions* $\{g_v\}_{v \in V}$ and a suite of *forwarding functions* $\{f_v\}$, one each for every node $v$ in the network.

Let $E_v$ be the set of edges adjacent to the node $v$. The stack-making function $g_v : V \to (E_v \times \Sigma^*)$ is evaluated at each ingress router $v$: it takes the destination address $u$ of the packet and outputs the outgoing edge $e = (v, w) \in E_v$ to which $v$ should send the packet. Further, it outputs a stack $g_v(u)$ that subsequently would take the packet from $w$ to $u$. The forwarding function $f_v : E_v \times \Sigma \to (E_v \times \Sigma^*)$, where $E_v$ is the set of edges incident to $v$, prescribes the behavior of the router. On obtaining a packet with stack $S$, the router $v$ performs the following actions:

- The router first pops the top label off the top of the stack; this label is denoted by $\ell$. A correct protocol must ensure that if the stack is empty, then $v$ is the packet's destination.
- If the packet arrived on edge $e$, let $f_v(e, \ell) = (e', \sigma)$ for $e' \in E_v$, $\sigma \in \Sigma^*$. The router now pushes $\sigma$ on top of the stack and sends the packet out on the edge $e'$.

If there is a single function $f$ such that $f_v = f$ for all $v$, the protocol $\mathcal{A}$ is called *uniform*; otherwise it is *nonuniform*. A protocol $\mathcal{A}$ is an $(L, s)$-protocol for a graph $G$ if it uses $L$ labels and has a maximum stack depth of $s$. We will mostly consider protocols that route on shortest paths, and this is implicitly assumed in our results. We will explicitly specify the cases in which packets may travel on nonshortest paths; a routing protocol has stretch $D$ if, given any pair $u, v \in V$, the protocol routes from $u$ to $v$ on a path with length at most $D$ times the length of the shortest path between $u$ and $v$.



$$\text{\textit{For all} } v\text{:}\quad f_v(\textit{left}, 0) = (\textit{right}, \langle\,\rangle)$$
$$f_v(\textit{left}, 1) = (\textit{right}, \langle 0\,0\rangle)$$

FIG. 1.1. *An example of MPLS routing.*

A simple example is given in Figure 1.1 with $\Sigma = \{0, 1\}$. All functions $f_v$ are the same, and only the relevant subset of the actions is shown here. The packet in the example is destined for the vertex labeled 10. Each node bases its decision on the top (shaded) label and the incoming edge.

**1.2. Our results.** The first routing problem that we consider is on the path $P_n$; here we show a substantial gap between uniform and nonuniform protocols. In particular, we show that while uniform protocols on the line with $L$ labels require a stack depth of $s = \Theta(Ln^{1/L})$, there are nonuniform protocols using $L$ labels that require a stack depth of just $O(\log_L n)$. Recall that there is an information-theoretic lower bound of $\log_L n$ on the stack depth, and hence the latter result is within a constant factor of the optimum.

The protocol for the path serves as our basic building-block when we consider arbitrary trees; we use it in conjunction with the so-called *caterpillar decomposition* [23] of trees into paths to get a $(\Delta + k, kn^{1/k}\log n)$ uniform protocol and a $\left(\Delta + k, \frac{\log^2 n}{\log k}\right)$ nonuniform protocol. (The additive $\Delta$ in the number of labels is unavoidable while routing on trees; if the maximum degree of a tree is $\Delta$, then we require at least $\Delta - 1$ distinct labels to achieve shortest path routing.) We also show that our uniform protocol is close to the best uniform protocol by give an almost matching lower bound when $k$ is $O(\log n)$. Furthermore, note that setting $k = \log n$ in the above nonuniform protocol gives a stack depth of $O(\log^2 n/\log\log n)$ with $\Delta + O(\log n)$ labels; we go on to refine the protocol and give a nonuniform $(\Delta + \log\log n, \log n)$ protocol as well.

The protocols developed for trees are then used to give routing protocols for general graphs. To this end, a *tree cover* of a graph $G = (V, E)$ is defined to be a family of subtrees $\mathcal{F}$ of the graph such that for each pair of vertices $u, v \in V$ there exists a tree $T \in \mathcal{F}$ which contains an (almost-)shortest path between $u$ and $v$. (See Definition 4.1 for a formal definition.) If the network has a tree cover with $t$ trees and we want to route a packet from $u$ to $v$, we can identify the appropriate tree for this pair of vertices and use the tree routing protocol to route on it; note that this causes the number of labels to increase by a factor of $t$, since the label has to encode the identity of the tree.

A simple argument can show that general graphs do not have $(\log n)$-sized tree covers unless the trees are allowed to stretch distances by $\Omega(\log n)$. Since a nonconstant stretch is often inadmissible in routing applications, we restrict our attention to important special classes of graphs. Our first result in this realm shows that graph families with $r(n)$-sized balanced vertex separators have $O(r(n)\log n)$-sized tree covers (without any stretch); using the same idea and the fact that planar graphs have balanced separators of size $O(\sqrt{n})$, we can get $O(\sqrt{n})$-sized tree covers for planar graphs. We then show a matching lower bound of $\Omega(\sqrt{n})$ for tree covers of planar graphs.

Though these matching results seem disheartening, we show that allowing tree covers to have a small stretch (of 3) makes them very powerful: we show that all planar graphs have $O(\log n)$-sized tree covers with stretch 3. The proof of this result uses the planar separator theorem of Lipton and Tarjan [21] in a novel way.

As the above discussion indicates, our protocols are extremely modular in nature; hence improvements in MPLS routing strategies for (say) trees will result in improvements for trees and graphs (see the paper of Gupta, Kumar, and Thorup [19] for such an improvement). Finally, we would like to emphasize that the constants involved in our protocols are small (and perhaps can be improved), which makes these useful in practice.

## 1.3. Previous work.

*Distributed packet routing protocols.* These protocols have been widely studied in the theoretical computer science community; see, e.g., [13, 14, 29, 28, 10] or the survey by Gavoille [15] on some of the issues and techniques. The results in these works are incomparable to our results, since the objectives of the two lines of research are quite different. Much of the conventional packet routing literature focuses on reducing the sizes of the routing tables and the sizes of the packet headers while performing near-shortest path routing. On the other hand, our work on MPLS routing may require more memory for setting up the initial stack than conventional routing protocols; however, once the stack is set up, the memory needed by each router just to forward the packets is very small. As a concrete example, the best result known for minimizing the total memory (i.e., summed over all the routers) on planar networks in traditional routing is $\tilde{O}(n^{4/3})$ due to Frederickson and Janardan [14]; in contrast, setting up the stack in our protocols requires more memory, but the total memory required for implementing the packet forwarding functions $f_v$ is $\tilde{O}(n)$.

Another difference with previous packet routing results is that our algorithms are name-independent; while many of the previous results require that the routing protocol be allowed to assign the names to the nodes in some convenient fashion, we do not make any such assumptions. (See the papers [4] and [2] for two other conventional routing schemes which are name-independent.)

*Spanners.* There is considerable literature on finding sparse *spanners* of graphs [1, 8]; these are sparse subgraphs that preserve distances well. However, results about spanners are interesting only when the original graph is not sparse, whereas the routing problems we address are nontrivial even for bounded degree graphs.

*Distance labeling schemes.* Another related corpus of work studies the problem of distance labeling of graphs [34, 26, 16]. The *distance labeling problem* requires assigning "short" labels to vertices so that the distance between two vertices can be inferred from their labels alone, without any additional information about the graph. The techniques used in many papers on distance labeling problems are similar to those we use, and involve finding good separators of graphs. However, the scope of the two problems are quite different. Indeed, the distance labeling problem precludes any knowledge of the global structure of the graph, and hence the label sizes are usually in the range of $\Theta(\log n)$. In contrast, MPLS routing schemes assume that the graph structure is known; the challenge in this case is in devising routing algorithms that base their routing decisions on a sublogarithmic (even possibly a constant) number of bits. As an example, MPLS routing for the path graph is quite nontrivial, whereas the distance labeling problem is trivial for the path. On the other hand, the similarity in the techniques allows us to use some of the results on MPLS routing to improve known results on distance labeling schemes. In section 5.2, we exhibit stretch-3 distance labeling schemes for planar graphs that use labels with $O(\log^2 n)$ bits; previous labeling schemes for planar graphs used a polynomial number of bits, even when a constant stretch was allowed [16].

*Tree covers.* The notion of tree covers was introduced in the papers of Awerbuch and Peleg [6] and Awerbuch, Kutten, and Peleg [5], though it had a slightly different definition. In those papers, the trees were not required to be spanning trees, and there was no explicit bound placed on the number of trees in the cover; however, the number of trees that contained any particular vertex had to be small. The objective was the same as in this paper: to construct a family of trees such that for every pair of vertices there was a tree containing an almost-shortest path between them. The best known constructions are due to Thorup and Zwick [33]; their algorithms find stretch $2k-1$ tree covers for general graphs, where each vertex lies in $\tilde{O}(n^{1/k})$ trees. This can be used to give stretch $2k-1$ MPLS routing schemes for arbitrary graphs with $\tilde{O}(n^{1/k})$ labels and polylogarithmic stack depth. In contrast to these results, the focus of our paper is on cases where constant stretch routing can be achieved with both stack depth and labels being $(\log n)^{O(1)}$.

**1.4. Organization.** The organization of the rest of the paper follows the above discussions: in section 2, we give MPLS routing protocols for the line, which highlight the difference in power between uniform and nonuniform routing. Section 3 gives the routing protocols for trees using the results for the line. Constructions of tree covers for graphs with small separators are given in section 4, which imply routing protocols for these graphs. Finally, constructions of stretch-3 tree covers for planar graphs, as well as their application to distance labeling schemes, appear in section 5.

## 2. Routing on the line.

In this section, we give uniform and nonuniform shortest path MPLS routing schemes for the path graph $P_n$; these will be used as basic building blocks for tree routing schemes in the next section. For our uniform scheme, we show that the maximum stack depth is $O(Ln^{1/L})$ when $L$ labels are used; our nonuniform scheme uses a maximum stack depth of $s = O(\log_L n)$. We also show that both these bounds on the stack depth are within constant factors of optimum.

**2.1. Uniform protocols.** The upper bound of $O(Ln^{1/L})$ is achieved by the following simple strategy. Suppose we wish to send a packet from a node $v$ to a node $u$ which is at distance $D$ from it. Suppose that $D-1$ can be written as $d_L d_{L-1} \dots d_1$ in base $n^{1/L}$. We push $d_i$ copies of $i$ onto the stack for $i$ going down from $L$ to 1. Now the forwarding function is trivial: nothing is pushed onto the stack when a 1 is seen; seeing an $i > 1$ causes $n^{1/L}$ copies of $(i-1)$ to be pushed onto the stack. In all cases, the outgoing edge is the one opposite the incoming edge.

A simple inductive argument can be used to prove that this forwarding function maintains the following invariant—when the packet is at a vertex $w$ at distance $D'$ from the destination $u$, the stack encodes $D'$ in the fashion described above. This immediately implies that the maximum stack depth is at most $Ln^{1/L}$; it also proves the correctness of the protocol, since the stack being empty implies $D' = 0$ and that the packet is indeed at the destination $u$. The following theorem follows from the discussions above.

THEOREM 2.1. *There is a uniform routing protocol for the $n$-vertex path with a maximum stack depth of $Ln^{1/L}$.*

This construction is indeed tight up to constant factors, as the following theorem shows.

THEOREM 2.2. *Any uniform routing protocol for the $n$-vertex path requires a stack depth of $\Omega(Ln^{1/L})$.*

*Proof.* We will show the above lower bound for the special case where all packets are forwarded on the path from left to right. Consider a directed auxiliary graph $H$ with the $L$ labels as its vertices, with an arc $(j, i)$ in the graph if the function $f$, on seeing $i$ on top of the stack, causes $j$ (among others) to be pushed on the stack. Note that any label that lies on a directed cycle in $H$ can never be used in the routing protocol, since the stack can never empty if this label appears on the stack. Hence we can assume that $H$ is a directed acyclic graph, and hence defines a partial order on the nodes.

Consider a total order consistent with the partial order implied by $H$; w.l.o.g., this is the order $1, 2, \dots, L$. Hence each label $i$ just corresponds to placing some number of labels $1, \dots, i-1$ on the stack. Therefore, the ordering of the labels on the stack does not matter—two stacks which have the same number of copies of label $i$ for all possible values of $i$ will reach the same destination.

Let $k_i$ be the number of copies of label $i$ on the stack, and hence $k_1+k_2+\cdots+k_L \leq s$. The number of solutions to this equation (and hence the number of distinct stacks) is $\binom{s+L}{L}$, which must be at least $n$, the number of possible destinations. Some simple algebra implies that $s = \Omega(Ln^{1/L})$, completing the proof. ∎

**2.2. Nonuniform protocols.** Interestingly, in the case for nonuniform protocols, the relationship between $s$ and $L$ almost achieves the information-theoretic bound of $s \geq \log_L n$. Since the direction of travel of the packet is decided by the edge at which it enters the vertex, it suffices to give a procedure to send packets from left to right.

For simplicity, consider the case when $L = 2$. Let the vertices on the path $P_n$ be numbered $0, 1, \dots, n-1$; this is only for ease of exposition, and the protocol does not depend on this labeling. We direct all edges in $P_n$ from left to right, and assign label 1 to these edges. We then add some (virtual) arcs $E'$ to this graph, also directed from left to right, and assign label 2 to these edges. It can be shown that these edges can be added so that the following properties are satisfied:

- *Single outgoing edge property*: Each vertex $v$ has at most one edge in $E'$ out of it.
- *Low-diameter property*: For any two vertices $u < v$, there is a directed path in $P_n \cup E'$ from $u$ to $v$ of length at most $3 \log n$.
- *Nesting property*: Let $u < u' < u''$ be three distinct vertices on the line. If $(u, u'')$ and $(u', v')$ are two directed edges in $E'$, then $v'$ does not lie to the right of $u''$; i.e., $v' \leq u''$. Essentially, no two edges in $E'$ *cross* each other; either they span disjoint portions of the line, or the span of one is contained within the span of the other.

One way to get such graphs is recursive: to build a graph $G_{2^k}$ on $2^k$ nodes, we take 2 copies of $G_{2^{k-1}}$ on $2^{k-1}$ nodes and attach them in series, giving a graph on $2^k - 1$ vertices. (A graph on 2 nodes is just a single arc.) A new vertex is now attached to the leftmost vertex by an arc labeled 1, and to the rightmost vertex by an arc labeled 2. This new vertex becomes vertex 0 in the new graph $G_{2^k}$, and the other vertices get suitably renumbered. In general, a graph $G_n$ on $n$ nodes is obtained by

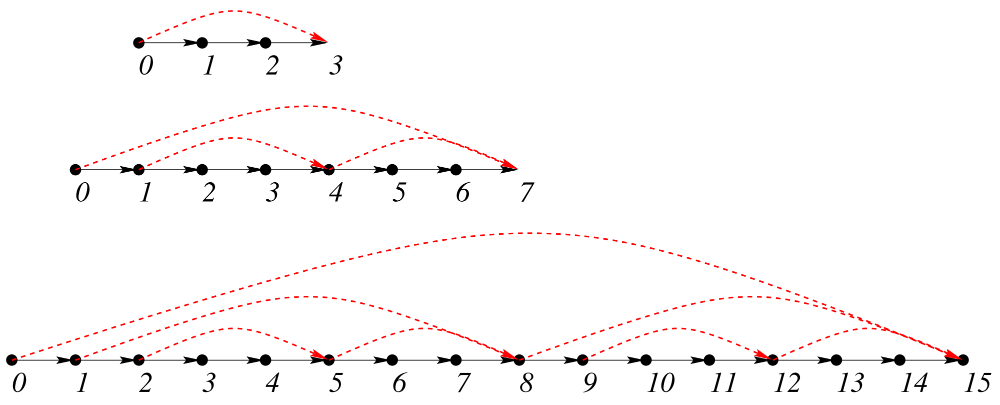


FIG. 2.1. *Specifying the routing protocol for the path $P_{16}$.*

taking a graph on $2^{\lceil \log_2 n \rceil}$ nodes and retaining only the leftmost $n$ nodes. An example for $n = 16$ is shown in Figure 2.1, where the solid edges are labeled 1 and the dotted edges are labeled 2.

The nesting property implies the following property for shortest paths in $P_n \cup E'$ (measured in terms of the number of hops).

LEMMA 2.3. *Let $u < u' < v' < v$ be four distinct nodes on the line $P_n$. If the shortest path $P$ from $u$ to $v$ in $P_n \cup E'$ contains $v'$, then the shortest path from $u'$ to $v$ contains $v'$.*

*Proof.* Suppose that the shortest path $P'$ from $u'$ to $v$ does not contain $v'$. Let $e = (w, w')$ be an edge in $P'$ such that $w < v' < w'$; clearly, such an edge must exist. We claim that $P$ must contain $w$. If not, since $u < w < v$, $P$ must contain an edge $e' = (x, x')$ such that $x < w < x'$; furthermore, $x' < v'$ since $P$ contains $v'$. But now $e$ and $e'$ violate the nesting property, a contradiction; hence $w \in P$.

Also, the portion of $P$ from $w$ to $v$ must be the shortest path from $w$ to $v$. Since we know that $P'$ contains $w$, replacing the portion of $P'$ after $w$ by the corresponding portion of $P$, we again get a shortest path $P''$ from $u'$ to $v$ which contains $v'$. Noting that there is a unique shortest path between any two vertices in $P \cup E'$ gives us a contradiction and proves the lemma. ◻

We can now describe the routing protocol. Given a node $u$ and a stack of labels $l_0, \ldots, l_r$ ($l_0$ being on the top), we define the *path defined by the stack* to be the sequence of edges obtained by starting from $u$ and following the edges labeled $l_0, \ldots, l_r$ in $P_n \cup E'$. If $u$ wants to send a packet to node $v$ ($u < v$), the stack is initialized so that the path defined by the stack is the shortest path from $u + 1$ to $v$, and the packet is sent to $u + 1$. The protocol will maintain the invariant that when a node $u'$ receives the packet, the path defined by the stack at that point is the shortest path from $u'$ to $v$. The low-diameter property thus ensures that the stack depth is at most $3 \log n$.

Let us see how to maintain the invariant. Let the packet be at $u'$, and let the edges labeled 1 and 2 originating from $u'$ be $e_1 = (u', u_1) \in P_n$ and $e_2 = (u', u_2) \in E'$, respectively. (If there is no label 2 edge from $u'$, the argument gets even simpler.) Since the edges in $E'$ do not exist, a packet can be forwarded along $e_1$ but not along $e_2$. If the top of the stack contains label 1, then $u'$ simply pops this label and sends the packet to $u_1$, the next vertex on $P_n$. Since the path defined by the stack was inductively the shortest path from $u'$ to $v$ and contained $u_1$, the new stack must define the shortest path from $u_1$ to $v$.

In the other case, the top of the stack has a 2: in this case, $u'$ pops it and pushes a set of labels which encode a shortest path from $u_1$ to $u_2$. Lemma 2.3 ensures that the shortest path from $u_1$ to $v$ contains $u_2$ as an intermediate node, and hence the path defined by the stack when it reaches $u_1$ is also a shortest path from $u_1$ to $v$, maintaining the invariant.

For the graph $G_n$ defined above, implementing the above protocol is extremely simple. As mentioned above, each router pops off a 1 if it sees one on top of the stack; the difference is in the handling of the 2's. If the router has outdegree 1, it just pops off the 2 (in fact, such a vertex will never see a 2); if it has outdegree 2, it replaces the 2 by two 2's. The following theorem follows from the above discussion.

THEOREM 2.4. *There is a nonuniform protocol for routing on the $n$-vertex path which uses* 2 *labels and stack depth at most* $3 \log n$.

It is trivial to encode $O(\log L)$ successive labels on the stack in a single label of size $L$, and hence we can use the above protocol to get the following theorem.

THEOREM 2.5. *There is a nonuniform protocol for routing on the $n$-vertex path which uses $L$ labels and stack depth at most $O(\log_L n)$. This is within a constant factor of the information-theoretic bound.*

**3. An algorithm for trees.** In this section, we consider the problem of routing on trees. Since we already have developed protocols for the line that are within constants of the best possible, we first show how to use them in a modular way to get protocols for trees; these are then refined to get better trade-offs.

Let the tree be $T$, and let it be rooted at $r$. All the algorithms use the so-called caterpillar decomposition of a tree into edge-disjoint paths. The *caterpillar dimension* [23] of a rooted tree $T$, henceforth denoted by $\kappa(T)$, is defined thus: for a tree with a single vertex, $\kappa(T) = 0$. Else, $\kappa(T) \leq k+1$ if there exist paths $P_1, P_2, \ldots, P_t$ beginning at the root and pairwise edge-disjoint such that each component $T_j$ of $T - E(P_1) - E(P_2) - \cdots - E(P_t)$ has $\kappa(T_j) \leq k$, where $T - E(P_1) - E(P_2) - \cdots - E(P_t)$ denotes the tree $T$ with the edges of the $P_i$'s removed, and the components $T_j$ are rooted at the unique vertex lying on some $P_i$. The collection of edge-disjoint paths in the above recursive definition form a partition of $E$, and are called the *caterpillar decomposition* of $T$.

By the very definition of caterpillar decompositions, it follows that the unique path between any two vertices of $T$ intersects no more than $2\kappa(T)$ of the paths in the decomposition. It can also be shown that $\kappa(T)$ is at most $\log n$ (see, e.g., [23]).

Now given a packet that has to traverse $k \leq 2 \log n$ paths, the stack just specifies the $k$ different stacks for routing on these paths, with each consecutive pair of stacks separated by one of $\Delta - 1$ special labels that specify the path to switch to. Hence, given an $(L, s)$ routing protocol for the line, we get a $(\Delta(T) + L - 1, s \times 2\kappa(T))$ protocol for the tree. Plugging in the values from the Theorems 2.1 and 2.5, we get the following result.

THEOREM 3.1. *Given a tree $T$ with maximum degree $\Delta$, there exists a $(\Delta + k - 1, 2\, k\, n^{1/k}\, \kappa(T))$ uniform routing protocol and a $(\Delta + k - 1, 6\, \log_k n\, \kappa(T))$ nonuniform routing protocol for $T$.*

Note that for $k = 2$ we have a $(\Delta + 1, 6 \log^2 n)$ nonuniform protocol, and for $k = \log n$ and constant $\Delta$ the worst-case guarantees for both these algorithms are approximately $(\log n + O(1), O(\log^2 n))$. The results of section 3.2 will show how to improve on this result in the nonuniform case. But before that, let us prove some lower bounds on uniform protocols for trees.

**3.1. A lower bound for uniform protocols on trees.** Before proving the lower bound, let us pin down what uniform protocols mean in the context of trees; since vertices may have varying degrees, they cannot have exactly the same actions. Hence, let us assume that there are $\Delta$ special labels belonging to the set $\Sigma_\Delta \subseteq \Sigma$, which are used only to travel a distance of one hop from a vertex, essentially by specifying which of the outgoing edges to take. Let $\Sigma' = \Sigma \setminus \Sigma_\Delta$ be the remaining labels. For each edge $e = \{u, v\}$, the vertex $v$ specifies another edge $e' = \{v, w\}$, called the *exit edge* for $v$ along edge $e$, such that any packet arriving at $v$ on edge $e$ having a label from $\Sigma'$ on top of the stack is forwarded only along $e'$. Furthermore, the action of all vertices on seeing a label from $\Sigma'$ is identical: i.e., an identical sequence of labels is placed on top of the stack, and then the packet is sent along the exit edge for the current vertex.

For such protocols, we now prove a lower bound that almost matches the result of Theorem 3.1 for the case of $k = \log n$.

THEOREM 3.2. *There exists a binary tree $\widehat{T}$ for which any uniform routing protocol that routes along shortest paths with $L = O(\log n)$ labels requires a stack depth of $\Omega(\log^2 n/\log\log n)$.*

*Proof.* Let us study some of the properties of uniform routing protocols on binary trees before we give the concrete instance for the lower bound. Look at any binary tree $T$; since each vertex has degree at most 3, it is easy to see that $\Sigma_\Delta$ needs to only contain a single label $\{l_\Delta\}$. Indeed, when a node $v$ receives a packet along an edge $e$, it can only send it on one of the other two edges, or else the routing would not be along a shortest path. One of these edges is the exit edge for $v$ along edge $e$, and hence we need only one label in $\Sigma_\Delta$.

Two vertices $u, v$ in $T$ are said to be connected by a *straight path* if all the internal vertices in the unique path connecting $u$ and $v$ have degree 2. Given two nodes $u, v \in T$, let $S[u, v]$ denote the stack depth needed to route a packet from $u$ to $v$. Given a label $l$, define $S_l[u, v]$ as the stack depth needed to route a packet from $u$ to $v$ such that when the packet reaches $v$, the top of the stack contains the label $l$. Since we may refer to several different protocols, we will use the notation $S_l[u, v](\mathcal{A})$ and $S[u, v](\mathcal{A})$ when talking about the protocol $\mathcal{A}$. The following lemma follows from the definition of a uniform protocol.

LEMMA 3.3. *Let $v \in T$ be a node of degree 3 and let $C_1, C_2, C_3$ be the components of $T \setminus \{v\}$. Let $v_i$ be the neighbor of $v$ in $C_i$. Then there exists a $j \in \{2, 3\}$ such that given any $x_1 \in C_1$ and $x_j \in C_j$, $S[x_1, x_j] \geq S_{l_\Delta}[x_1, v] + S[v, x_j] - 1$.*

*Proof.* Consider the edge $e = \{v_1, v\}$. Suppose the exit edge for the vertex $v$ along edge $e$ is the edge $\{v, v_2\}$. Now if we want to send a packet from $x_1$ to $x_3$, it must contain $l_\Delta$ on top of stack when it reaches $v$. Hence the part of this stack which takes the packet from $x_1$ to $v$ contributes to $S_{l_\Delta}[x_1, v]$. The part of the stack below $l_\Delta$ can actually route from $v_3$ to $x_3$, and hence $S[v, x_3] \leq S[v_3, x_3] + 1$. Since $S[x_1, x_3] = S_{l_\Delta}[x_1, v] + S[v_3, x_3]$, the lemma follows. ◻

Given two protocols $\mathcal{A}''$ and $\mathcal{A}'$, we say that $\mathcal{A}'$ *strictly dominates* $\mathcal{A}''$ if for every pair of vertices $u, v$ and label $l \in \Sigma$, the following hold true: (i) $S_l[u, v](\mathcal{A}') \leq S_l[u, v](\mathcal{A}'')$, (ii) $S[u, v](\mathcal{A}') \leq S[u, v](\mathcal{A}'')$, and (iii) there is a pair $u, v$ of vertices, and some label $l$ such that $S_l[u, v](\mathcal{A}') < S_l[u, v](\mathcal{A}'')$. Fix a uniform routing protocol $\mathcal{A}$ that is not strictly dominated by any other $\mathcal{A}'$.

LEMMA 3.4. *Let $T$ contain a straight path of length $n'$ joining vertices $u$ and $v$. There exists a value $x$ with $n'/2 \leq x \leq n'$, such that if $u', v'$ are any two vertices in $T$ connected by a straight path of length $x$, then $S_{l_\Delta}[u', v']$ is $\Omega(\log n'/\log\log n')$.*

*Proof.* Let $P$ be the path joining $u$ and $v$, and let $V'$ be the vertices in $P$ whose distance from $u$ lies between $n'/2$ and $n'$. We claim that there is a vertex $w \in V'$ such that $S_{l_\Delta}[u, w]$ is $s' = \Omega(\log n'/\log\log n)$. Indeed, if we are allowed a stack depth of $s'$, the number of different stacks possible is $L^{s'}$. Since this must be at least $n'/2$, it follows that $s'$ is at least $\Omega(\log_L n') = \Omega(\log n'/\log\log n)$.

Let the distance between $u$ and $w$ be $x$, suppose that $u'$ and $v'$ are two vertices with a straight path of length $x$ between them, and suppose $S_{l_\Delta}[u', v'] < s'$. Then the uniformity of $\mathcal{A}$ implies that, keeping other things the same, we can make $S_{l_\Delta}[u, w] < s'$ and get a protocol that strictly dominates $\mathcal{A}$, a contradiction. This proves the lemma. ∎

For each integer $x = n^{1/3}, \ldots, 2n^{1/3}$, let $T_x$ denote the complete binary tree of depth $1/6 \log n$ and having $x$ subdivisions on each edge. Let $\widehat{F} = \cup T_x$ be the forest formed by the union of all these trees; this is the lower bound instance. (We can extend $\widehat{F}$ to a single binary tree $\widehat{T}$ by adding suitable edges; the bound clearly holds for $\widehat{T}$ as well.) Let us define a *branching node* in $T_x$ to be a node of degree 3.

Note that $\widehat{F}$ (and thus $\widehat{T}$) has a straight path of length $2n^{1/3}$ between two vertices. Hence, by Lemma 3.4, there is a value $x$ satisfying $n^{1/3} \le x \le 2n^{1/3}$ such that if $u, v$ are two *branching* nodes in $T_x$ joined by a straight path, then $S_{l_\Delta}[u, v]$ is $\Omega(\log n'/\log\log n) = \Omega(\log n/\log\log n)$. Now, using Lemma 3.3 repeatedly, we can demonstrate a path from the root to a leaf $y$ of $T_x$ such that routing from the root of $T_x$ to $y$ requires stack depth $\Omega(\log^2 n/\log\log n)$. ∎

**3.2. Improved nonuniform protocols.** In this section, we show how to obtain nonuniform routing protocols which require a stack depth of only $O(\log n)$ (as in the case of the path graph), but with $\Delta + O(\log\log n)$ labels.

THEOREM 3.5. *There exists a* $(\Delta + 2\log\log n, 21\log n)$ *nonuniform routing protocol for trees.*

*Proof.* For any vertex $w \in T$, let $T_w$ be the subtree rooted at $w$. To prove the theorem, we will need the following claim, which will be proved by induction on $|T_w|$.

CLAIM 3.6. *There exists a nonuniform protocol to route a packet from any vertex* $w$ *to a descendant* $x$ *in* $T_w$ *using* $2\log k + (\Delta - 1)$ *labels and a stack of depth at most* $18\,k$*, where* $k = \lceil \log_2 |T_w| \rceil$.

To use this protocol to route packets from any vertex $u$ to any other vertex $x$, we first route it to $w$, the least common ancestor of $u$ and $x$, and then use the protocol from Claim 3.6 to route from $w$ to $x$. Sending the packet from $u$ to $w$ is very simple; indeed, the problem of routing a packet from a vertex to some ancestor in a tree is isomorphic to the problem of routing on the line. This can be done by the protocol in Theorem 2.5 with 2 labels and a stack depth of $3\log n$; Claim 3.6 then ensures that the stack depth to route from $w$ to $x$ is at most $18\log_2 n$. ∎

*Proof of Claim* 3.6. Let $P'_1, \ldots, P'_t$ be the paths in the caterpillar decomposition that contain the vertex $w$, and let $P_i$ be the subpath of $P'_i$ that contains $w$ and its descendants; i.e., $P_i = P'_i \cap T_w$.

We make an additional requirement on the caterpillar decomposition for the tree $T$, which is the following *halving* property: we demand that for a vertex $v \in P_i$, any connected component of $T_w - \{v\}$ not containing a node of $P_i$ has at most $\lfloor |T_w|/2 \rfloor$ nodes. It can be shown that such a caterpillar decomposition exists (see, e.g., [23]).

The basic idea is similar to the one used in section 3; we will route on a series of paths, using the protocol for the line given in Theorem 2.5 as the basic building block. As before, when we switch between paths, a special subset of $\Delta - 1$ of the

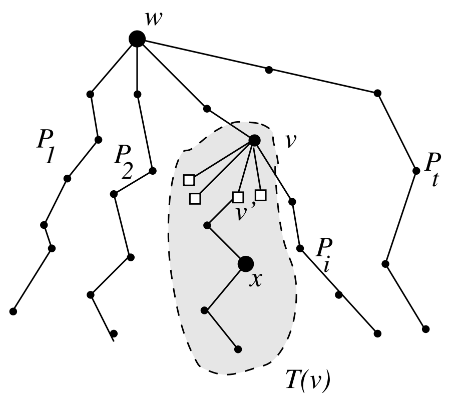


FIG. 3.1. *The tree $T_w$ in the proof of Claim* 3.6. *Vertices marked by squares belong to $V'$.*

labels will be used to indicate which of the new paths to continue on. The rest of the proof indicates how the other $2\log k$ labels can be used for the rest of the routing.

The base case of $|T_w| = 1$ is trivial. We now show how to route a packet from $w$ to $x$, where $x$ is a descendant of some node in $P_i - \{w\}$; the lemma follows from the fact that the paths $P_1, \ldots, P_t$ intersect only at the origin node $w$.

Consider a vertex $v \in P_i - \{w\}$, and let $V'$ be the immediate children of $v$ which do not lie on $P_i$ itself (for an illustration, see Figure 3.1). Define $T(v)$ be a subtree rooted at $v$ containing $v$, the subset of $v$'s children $V'$ that do not lie on $P_i$, and all the descendants of $V'$. Observe that if $v_1 \neq v_2 \in P_i$, then $T(v_1)$ and $T(v_2)$ are disjoint. We define $t(v)$, the *index* of a node $v$, to be $\lceil \log_2 |T(v)| \rceil$. Let the *group* $I(j)$ be the set of nodes in $P_i - \{w\}$ with index $j$. Note that if $t(v) = j$, then $|T(v)| \geq 2^{j-1}$; since all the trees $T(v)$ are disjoint and their union contains at most $|T_w| \leq 2^k$ nodes, there can be at most $2^{k-j+1}$ nodes in $I(j)$.

We now define $\log k$ *supergroups*, with each supergroup being the union of several groups $I(j)$. For each $p = 0, \ldots, \log k$, define

$$\mathcal{I}(p) = \bigcup_{q=2^p}^{2^{p+1}-1} I(k-q+1).$$

Since the size of $\mathcal{I}(p)$ is maximized when all nodes in it come from the group $I(k - 2^{p+1} + 2)$, the supergroup $\mathcal{I}(p)$ can contain at most $2^{2^{p+1}-1}$ nodes. We divide the $2\log k$ labels we have into $\log k$ sets $L_1, \ldots, L_{\log k}$, with each $L_i$ containing two labels. The labels in $L_p$ are used to route from $w$ to nodes that lie in $\mathcal{I}(p)$. If a node in $P_i$ that does not belong the supergroup $\mathcal{I}(p)$ sees a label in $L_p$ on top of the stack, it merely forwards the packet on to its unique child lying in $P_i$. Theorem 2.5 implies that we can use the labels in $L_p$ to route from $w$ to all nodes in $\mathcal{I}(p)$ using a stack depth of at most $3(2^{p+1} - 1)$.

Suppose that $w$ wants to send a packet to $x \in T(v)$, with $v \in I(j)$ and $I(j) \subseteq \mathcal{I}(p)$. The top of the stack contains the labels (belonging to $L_p$) which specify how to send the packet from $w$ to $v$; this requires a depth of at most $3(2^{p+1} - 1)$. Let $v' \in V'$ be the child of $v$ such that $x \in T_{v'}$; hence, the next label on the stack is one of the $\Delta - 1$ labels and causes $v$ to forward the packet to $v'$. The remaining part of the stack specifies how to route the packet from $v'$ to $u$; by induction, this part has depth at most $18\lceil \log |T_{v'}| \rceil$.

By the facts that $v \in \mathcal{I}(p)$ and $|T_{v'}| \le |T(v)|$, it must be the case that $\lceil \log |T_{v'}| \rceil \le k - 2^p + 1$. Furthermore, the halving property of the caterpillar decomposition ensures that $|T_{v'}| \le |T_w|/2 \le 2^{k-1}$. Hence the total stack depth to route from $w$ to $x$ is at most

$$\begin{aligned} & 3\,(2^{p+1}-1) + 1 + 18\,\lceil \log T_{v'} \rceil \\ \le\ & 3\,(2^{p+1}-1) + 1 + 6\,(k - 2^p + 1) + 12\,(k-1) \\ \le\ & 18\,k. \end{aligned}$$

This shows that we can achieve a stack depth of at most $18\,k$ using only $\Delta + 2\,k - 1$ labels, proving the claim. $\square$

**4. Covering graphs by trees.** Several problems arise in trying to extend the approach of decomposing arbitrary graphs into paths, and then using the path routing schemes on them: the shortest paths between vertices in general graphs are not unique, and they intersect in nontrivial ways, making it difficult to arrive at a useful notion of a path decomposition. On the other hand, if we find a family of $t$ subtrees, such that for each pair of vertices there was a tree in this family that maintained the shortest path distance between them, we could use this for routing. Indeed, we could use the tree routing scheme of the previous section, with the labels also specifying which of these $t$ trees we were routing on; this would cause the number of labels to increase by a factor of $t$. We could, in fact, relax the distance preservation condition and allow distances to be stretched by a small factor even in the best tree. Motivated by this, we define a *tree cover* of a graph, as follows.

DEFINITION 4.1. *Given a graph $G = (V, E)$, a* tree cover *(with* stretch *$D$) of $G$ is a family $\mathcal{F}$ of subtrees $\{T_1, T_2, \ldots, T_k\}$ of $G$ such that for every $u, v \in V$ there is a tree $T_i \in \mathcal{F}$ such that $d_{T_i}(u, v) \le D\, d_G(u, v)$.*

Note that, since the trees $T_i$ are subtrees of $G$, it immediately follows that $d_{T_i}(u, v) \ge d_G(u, v)$ for all $u, v \in V$ and $T_i \in \mathcal{F}$. The following theorem formalizes the discussion above.

THEOREM 4.2. *Given an $(L, s)$ protocol for routing on trees, and a tree cover $\mathcal{F}$ of $G$ with stretch $D$, there exists an $(L\,|\mathcal{F}|, s)$ protocol (with stretch $D$) for $G$.*

Tree covers have been previously defined and used for conventional routing applications in [6, 5] (see also [27, Chapter 15]). Note that our tree covers, as defined in Definition 4.1, are slightly different from those in the previous literature; we do not place a restriction on the number of trees in which a vertex appears, instead placing a uniform restriction on the number of trees in the family.

**4.1. Some lower bounds.** It is easy to see that the size of a tree cover may necessarily be large: if we require a stretch-1 tree cover for the complete graph $K_n$, the union of the trees $T_i$ in the cover must cover every edge, and hence $\Omega(n)$ trees are required in this case.

Even allowing stretch-$D$ tree covers does not help much: there are constructions of $n$-vertex graphs with $\Omega(n^{1+4/(3g-6)})$ edges which have girth $g$ [22]. For such a graph, desiring a stretch of at most $g - 2$ forces the union of $T_i$ to contain every edge of the graph. This gives a lower bound of $\Omega(n^{4/3D})$ on the size of tree covers having stretch $D$. For the special case of stretch 3, note that any stretch-3 tree cover for the complete bipartite graph must also require $\Omega(n)$ trees.

While the graphs considered above are not sparse, these lower bounds can be strengthened to obtain the following theorem.

THEOREM 4.3. *There are $n$-vertex graphs with maximum degree 3 for which any stretch-$D$ tree cover must contain $n^{\Omega(1/D)}$ trees.*

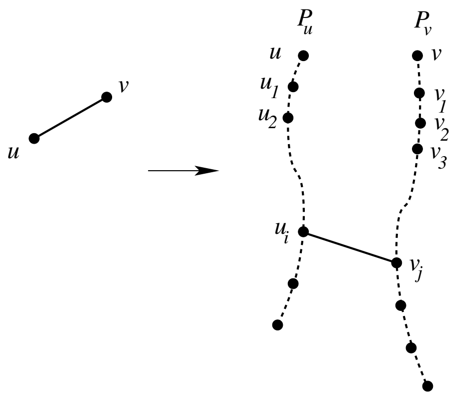


FIG. 4.1. *Mapping an edge of $G$ into an edge of $H$. The dotted edges have length* 0.

*Proof.* Let us consider an $\hat{n}$-vertex graph $G = (V, E)$ with girth $D + 2$ and $\hat{m} = |E| = \Omega(\hat{n}^{1+4/3D})$ edges given by Margulis [22], and create an $n$-vertex degree-3 graph $H = (V', E')$ from it as follows. Initially $V' = E' = \emptyset$. For each vertex $v \in V$ of degree $\delta(v)$, we create a path $P_v$ with $\delta(v)$ vertices, with $v$ as one of its endpoints, and consisting of $\delta(v) - 1$ other (new) vertices. We set the length of all these edges to be 0 and add all these vertices and edges to $H$. We now add some more edges to $E'$: for an edge $(u, v) \in E$, we add an edge (of length 1) between some vertex of $P_u$ and some vertex of $P_v$ such that the degree of all vertices stays at most 3. (Since there are $\delta(v)$ vertices in $P_v$ and $\delta(v)$ such edges are added, this is trivially possible.) It is easy to check that $n = |V'| = 2|E|$ and $|E'| \leq \frac{3}{2}|V'| = 3|E'|$. Furthermore, since $G$ is the minor of $H$ obtained by contracting all the 0-length edges, and the 0-length edges themselves induce a forest in $H$, it can be verified that each cycle on $H$ contains at least $(D + 2)$ edges of unit length.

Let $\mathcal{F}_H = \{T'_1, T'_2, \ldots, T'_t\}$ be a stretch-$D$ tree cover for $H$; we want to infer that $t = n^{\Omega(1/D)}$. To this end, we will use $\mathcal{F}_H$ to define a family $\mathcal{F}_G$ of $t$ forests in $G$ such that every edge in $G$ must be contained in some forest in $\mathcal{F}_G$. This would imply that $t \geq \hat{n}^{4/3D} \leq n^{\Omega(1/D)}$.

Given a tree $T'_i \in \mathcal{F}_H$, let us define the forest $F_i$ of $G$ as follows: consider an edge $(u, v) \in E$. Due to the construction above, this edge corresponds to an edge $(u_i, v_j) \in E'$ with $u_i \in P_u$ and $v_j \in P_v$. (See Figure 4.1 for an illustration.) We add $(u, v)$ to $F_i$ if the entire path $\langle u, u_1, u_2, \ldots, u_i, v_j, v_{j-1}, \ldots, v_1, v\rangle$ lies in $T'_i$. It is easily verified that $F_i$ is a forest.

Let us now prove that every edge in $G$ must be contained by some forest $F_j \in \mathcal{F}_G$ if $\mathcal{F}_H$ is a stretch-$D$ tree cover. Consider an $(u, v) \in E$ as above; we claim that one of these trees $T'_i \in \mathcal{F}_H$ must contain the entire path $\langle u, u_1, u_2, \ldots, u_i, v_j, v_{j-1}, \ldots, v_1, v\rangle$. Indeed, since every cycle in $H$ contains $(D + 2)$ edges of unit length, the absence of this path in $T'_i$ would imply that the alternative path between $u$ and $v$ in $T'_i$ would incur a stretch of at least $D + 1$. Let $T'_j$ contain this path; by the mapping above, $F_j$ now contains the edge $(u, v) \in E$. Hence each edge in $E$ is contained in some forest in $\mathcal{F}_G$, implying that $t \geq n^{4/(4+3D)}$ and proving the theorem. $\square$

In view of these negative results, we will focus our attention on some natural important families of graphs, like planar graphs and those with small separators. In this section, we consider tree covers with stretch 1 and show that graphs which have small separators (and whose subgraphs also have this property) have good tree covers.

This result also shows that planar graphs have $O(\sqrt{n})$-sized tree covers; we then show that this bound is tight.

**4.2. Unit-weighted grid.** To warm up, let us give the following simple result, which illustrates some of the ideas used later in the section.

PROPOSITION 4.4. *The unit-weighted $n$-vertex grid $G$ has tree covers of size $O(\log n)$.*

*Proof.* Let the vertex set of $G$ be $V = \{(i,j) \mid 1 \le i,j \le \sqrt{n}\}$. Consider the tree $T$ defined by taking the center vertical path $P = \{(\sqrt{n}/2, j) | 1 \le j \le \sqrt{n})\}$ and the horizontal paths $P_j = \{(i,j) | 1 \le i \le \sqrt{n})\}$ for all $j$. It is easy to check that, for any two vertices that lie in different halves of the grid defined by the vertical path $P$, the shortest path lies in $T$.

To maintain distances between vertices which lie on the same side of the vertical path, we can recurse on both the smaller grids $G_L$ and $G_R$ induced by $\{(i,j) \in V \mid i < \sqrt{n}/2\}$ and $\{(i,j) \mid i > \sqrt{n}/2\}$, respectively. (A construction identical to the one for the square grid works for rectangular grids as well, and so the recursion is well defined.) Inductively, we get two families of at most $t = \log(n/2)$ forests, one for each subgrid $G_L$ and $G_R$; let them be $\mathcal{F}_L = \{F_1', F_2', \ldots, F_t'\}$ and $\mathcal{F}_R = \{F_1'', F_2'', \ldots, F_t''\}$, respectively. Note that defining $F_i = F_i' \cup F_i''$ gives us $t$ forests of the original graph $G$ (since $F_i'$ and $F_i''$ are vertex disjoint), and each of these can be extended to a spanning tree by adding some more edges. Finally, adding the tree $T$ to these $\log n - 1$ trees gives us the desired $\log n$-sized tree cover. ∎

Combining this result for the grid with Theorems 4.2 and 3.5 gives us an $(O(\log n \log\log n), O(\log n))$ routing protocol. While it is not clear how to improve the tree cover of Proposition 4.4, it is indeed possible to get a better routing scheme for the unweighted grid. Given two vertices $u = (i,j)$ and $v = (i',j')$, there is a shortest path between them that goes from $u$ to $w = (i,j')$ and then from $w$ to $v$. The protocol specifies how much distance to go without changing the first coordinate, and then how far to go without changing the second coordinate; this gives us an $(O(1), O(\log n))$ routing scheme for the grid.

**4.3. Graphs with small separators.** We say that a graph $G$ on $n$ vertices admits *$r(n)$-sized hierarchical separators* if it can be separated into pieces of size at most $2n/3$ by removing at most $r(n)$ vertices, and any connected component $G_i$ thus obtained has a separator of size $r(|G_i|)$, and so on. Using some of the ideas from the construction above, we give tree covers of size $O(r(n) \log n)$ for families of graphs which admit $r(n)$-sized hierarchical separators. It is well known that for planar graphs, $r(n) = O(\sqrt{n})$ [21], and for treewidth-$k$ graphs, $r(n) = k$ (see, e.g., [7]). (We shall assume that $r(n)$ is a monotonically increasing function of $n$.)

The basic idea is simple: we first find a separator $S$ of $G$ of size at most $r(n)$. For each of the vertices $s \in S$, we take the shortest-path tree $T_s$ rooted at $S$. Lemma 4.5 now shows that the distance between a vertex in any component $C$ of $G - S$ and another vertex in $G - C$ is maintained by the tree $T_s$ for some $s \in S$.

LEMMA 4.5. *For any pair of vertices $u, v \in T$ for which the shortest path $P$ connecting them intersects $S$, there is a tree $T_s$ which contains the shortest path between $u$ and $v$.*

*Proof.* Let us assume, for the sake of convenience, that $P$ is the unique shortest path between $u$ and $v$. Let $P \cap S$ contain the vertex $s$. Then $P$ must be the concatenation of the shortest path from $s$ to $u$ and that from $s$ to $v$. Since both these paths lie in $T_s$, the claim is proved. ∎

We now have to maintain distances between vertices lying in some component of $G-S$. However, each of these components has size at most $2n/3$; recursively, for each component, we can find a tree cover of size $r(2n/3)\log_{3/2}(2n/3) \leq r(n)(\log_{3/2} n - 1)$. Finally, pairing them up and adding the set of $r(n)$ trees created at the top level, we get a tree cover with $r(n)\log_{3/2} n$ subtrees, proving the following theorem.

THEOREM 4.6. *Given a graph which admits $r(n)$-sized hierarchical separators, we can find a tree cover of size $O(r(n)\log n)$.*

## 4.4. Lower bounds.

For planar graphs, we can obtain tree covers of size $O(\sqrt{n})$; this requires using the existence of separators of size $r(n) = O(\sqrt{n})$ and being slightly more careful in the above analysis. We now show that this result for planar graphs is tight.

The lower bound is achieved on a reweighted grid: let $G = (V, E)$ be the $n = t \times t$ square grid, where the vertices are $V = \{(i, j) \,|\, 1 \leq i, j \leq t\}$. Since there may be several shortest paths between two vertices, let us break the symmetry. Define a partial order on the edges by declaring all vertical edges in the grid to be less than the horizontal edges; the lexicographically least shortest path between two vertices in the grid is now defined to be the (unique) shortest path between them. (It can be verified that there is a setting of edge weights that achieves this as well; e.g., set the length of an edge $e$ joining vertices $(i, j)$ and $(i', j')$ to be $1 + \epsilon\,(\min(i, i') + (1+\epsilon)\min(j, j'))$ for $\epsilon = 1/n$.) Defining this total order ensures that the lexicographically least path between two vertices consists of all the vertical edges followed by the horizontal edges. The following lemma then follows immediately.

LEMMA 4.7. *Given any two vertices in $G$, there is a unique shortest path between them. Furthermore, this shortest path has at most one bend.*

Let $T$ be any spanning tree of $G$, and let $\mathcal{S}_T$ be the set of pairs of vertices $(u, v) \in V \times V$ such that $T$ contains a shortest path between $u$ and $v$ as defined above. The following key lemma shows that $\mathcal{S}_T$ cannot be too large.

LEMMA 4.8. *For any spanning tree $T$ of $G$, the number of vertex pairs whose shortest paths lie in $T$ is $O(t^3)$; i.e., $|\mathcal{S}_T| = O(t^3)$.*

Before we prove the lemma, let us see what it implies: since there are $\binom{n}{2} = \Omega(t^4)$ pairs of vertices, this shows that we need $\Omega(t) = \Omega(\sqrt{n})$ trees in the cover, proving the following theorem.

THEOREM 4.9 (lower bound theorem). *There exist length assignments to the edges of the $n$-vertex grid so that any tree cover (with stretch 1) is of size $\Omega(\sqrt{n})$.*

*Proof of Lemma* 4.8. A connected path $P \subseteq T$ is called *straight* if it does not have any bends and is of maximal length; i.e., adding any other edge of $T$ to $P$ results in a bend. Let $\{P_1, \ldots, P_k\}$ be the set of all straight paths in $T$, and let $V_i = V(P_i)$. Clearly, each straight path has at most $t$ vertices, i.e., $|V_i| \leq t$. Furthermore, the paths must cover the entire tree, and hence $\cup_{i=1}^{k} V_i = V$. The maximality of these straight paths ensures that for any $i \neq j$, $|V_i \cap V_j| \leq 1$.

Construct the intersection graph $T' = (V', E')$ for these paths, where $V'$ has a vertex $p_i$ for each path $P_i$ and $E'$ contains an edge joining $p_i$ and $p_j$ if and only if $V_i \cap V_j \neq \emptyset$. Since a cycle in $T'$ would imply a cycle in $T$, it follows that $T'$ is a tree.

CLAIM 4.10. *Let $u \in V_i$, $v \in V_j$ be two vertices. The tree $T$ preserves the shortest path between $u$ and $v$ only if either $i = j$ or $(p_i, p_j)$ is an edge in $T'$.*

*Proof.* By Lemma 4.7, the shortest path $P$ between $u$ and $v$ has at most one bend. Suppose $E(P) \subseteq E(T)$; then $P$ either lies within some straight path $P_l$ (and hence $i = j = l$), or lies in the union of two straight paths which intersect (and hence $(p_i, p_j) \in E'$). $\square$

Let $t_i = |V_i| \leq t$, and define the *weight* of the tree $T'$ to be

$$f(T') = \sum_{p_i \in V'} t_i^2 + \sum_{(p_i,p_j)\in E'} (t_i - 1)(t_j - 1). \tag{4.1}$$

Claim 4.10 implies that $|\mathcal{S}_T| \leq f(T')$, and hence it suffices to show that $f(T')$ is $O(t^3)$.

For the rest of the proof, we will not look at the semantics of the sets again. Instead we will consider an arbitrary set system on $t^2$ vertices satisfying the following properties: (a) each set $V_i$ has size $t_i \leq t$, (b) two sets intersect in at most one element, and (c) the intersection graph of the subsets is a tree. For such an intersection tree $T'$, we assign weight $t_i^2$ to each node and $(t_i - 1)(t_j - 1)$ to each edge $(p_i, p_j)$ in $T'$; hence $f(T')$ is the total weight of vertices and edges in $T'$. The following claim now suffices.

CLAIM 4.11. *For an intersection tree $T'$ satisfying conditions* (a)–(c) *above, the total weight $f(T')$ of vertices and edges is $O(t^3)$.*

*Proof.* Let us root $T'$ at any vertex, and record the following useful lemma.

LEMMA 4.12. *Suppose $(p_i, p_j) \in E'$ with $t_i + t_j \leq t$. Then deleting the two sets $V_i$ and $V_j$ and adding $V_i \cup V_j$ to the set system maintains properties* (a)–(c) *and does not cause the weight to decrease.*

*Proof.* Clearly, the size of the new set $|V_i \cup V_j| = t_i + t_j - 1 \leq t$, satisfying property (a). Furthermore, since the intersection graph was initially a tree, no $V_l$ (for $l \notin \{i, j\}$) can intersect both of $V_i$ and $V_j$, and hence $|V_l \cap (V_i \cup V_j)| \leq 1$, satisfying (b). Also, the new intersection graph is obtained by contracting the edge $(p_i, p_j)$ in $T'$. Finally, the increase in weight of the tree is at least

$$\begin{aligned}(t_i + t_j - 1)^2 - t_i^2 - t_j^2 - (t_i - 1)(t_j - 1) &= 2t_it_j - 2t_i - 2t_j - t_it_j + t_i + t_j \\ &= (t_i - 1)(t_j - 1) - 1 \ \geq \ 0,\end{aligned}$$

hence proving the lemma. ☐

We can perform the above operation on the tree $T'$ up to the point at which every edge $(p_i, p_j) \in E'$ has $t_i + t_j \geq t + 1$. We call a leaf $p_i$ in the resulting tree $T'$ *bad* if $t_i < t/2$, and we delete all these leaves from $T'$ to get a new tree $T''$. Since the parent vertices of the deleted leaves must be of size at least $t/2$, all leaves in $T''$ (which are either surviving leaves of $T'$ or parents of leaves deleted from $T'$) have size at least $t/2$.

We claim that $T''$ has $4t$ nodes. Indeed, let us root $T''$ at some vertex and greedily match nodes in $T''$: we start at the root, which we match to one of its children. At the $i$th step, we look at the unmatched nodes at depth $i$ in $T''$, and match them to one of their children. At the end of the process, all nodes except some leaves of $T''$ would be matched. Note that, for each edge $(p_i, p_j)$ in the matching, $t_i + t_j \geq t + 1$, and hence there are at most $2 \cdot t^2/(t + 1) < 2t$ matched nodes. Furthermore each leaf of $T''$ that is unmatched has at least $t/2$ nodes, and hence $T''$ has at most $2t$ unmatched nodes; this proves the claim.

Let us now bound $f(T')$. The contribution of the edges of $T''$ is at most $t^2(4t-1)$, since each edge can contribute only $t^2$. The edges connecting the deleted bad nodes to their parents contribute $O(t^3)$; this is because the bad leaves are all disjoint, and hence $\sum t_i$ (summed over the bad leaves) is at most $t^2$. For the vertex contributions, note that $T''$ has $4t$ nodes, each having at most $t$ elements, and hence the contribution of these nodes is at most $O(t^3)$. Finally, the bad leaves are all disjoint with $\sum t_i \leq t^2$, and $\max_i t_i \leq t$; hence $\sum_i t_i^2 \leq (\sum_i t_i) \times (\max_i t_i) \leq t^3$. Summing up all these terms shows that $f(T') = O(t^3)$. ☐

Since $f(T')$ gave a bound on the number of pairs of vertices in the grid whose distances could be maintained by a single tree, we need at least $\Omega(t) = \Omega(\sqrt{n})$ trees in the cover, thus proving the theorem. ☐

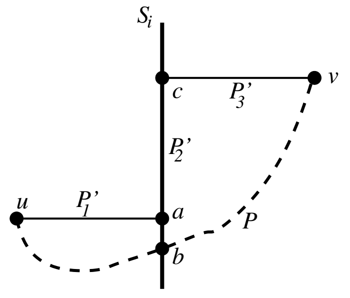


FIG. 5.1. *Path between $u$ and $v$ in the proof of Theorem* 5.1.

## 5. Tree covers for planar graphs.

In the previous section, we saw that planar graphs may require polynomially sized tree covers if we allow no stretch. The results of this section show that all planar graphs have stretch-3 tree covers of size $O(\log n)$. For this result, we use the planar separator theorem of Lipton and Tarjan [21] in an unusual way.

### 5.1. Isometric separators.

We can refine the ideas in section 4.3 to get an $O(\log n)$-sized stretch-3 tree cover for all planar graphs. Given a graph $G = (V, E)$, define a *k-part isometric separator* to be a family $\mathcal{S}$ of $k$ subtrees $S_1 = (V_1, E_1), \ldots, S_k = (V_k, E_k)$ of $G$ such that the following two conditions hold:

- $S = \cup_i V_i$ is a $\frac{1}{3}$–$\frac{2}{3}$ separator of $G$; i.e., each component in $G \setminus S$ has at most $\frac{2n}{3}$ vertices, where $n$ is the number of vertices in $G$.
- For each $i$ and each pair of vertices $u, v \in S_i$, $d_{S_i}(u, v) = d_G(u, v)$; i.e., each of the subtrees $S_i$ contains the shortest paths between its constituent vertices, and hence the tree metric on $S_i$ is isometric to the restriction of the shortest path metric of $G$ to $V_i$.

Note that the parameter of interest is not the total number of vertices in $S$, but only the number of isometric subtrees. Trivially, any graph having a $\frac{1}{3}$–$\frac{2}{3}$ separator of size $r(n)$ has an $r(n)$-part isometric separator, with each $S_i$ having just a single vertex. Isometric separators can also be used to give small-stretch routing; a simple extension of the ideas in the previous sections demonstrates the following theorem.

THEOREM 5.1. *Given a graph $G = (V, E)$ with $r(n)$-part isometric separators, there exists a stretch-3 tree cover with $O(r(n) \log n)$ trees.*

*Proof.* The algorithm is very similar to that of Theorem 4.6. Let $S_1, S_2, \ldots, S_{r(n)}$ be the subtrees for $G$. For each $i$, we construct a tree $T_i$ as follows: we contract the edges of $S_i$ to a new node $s_i$ and find a shortest-path tree from $s_i$ in the resulting graph. We then expand back the contracted edges in this tree, and call the resulting tree $T_i$. Note that $T_i$ contains $S_i$, as well as a shortest path from every vertex in $V - V_i$ to the subtree $S_i$. We claim that if the shortest path between any two vertices in $V$ intersects the separator vertices $\cup_i V_i$, then one of these trees approximately maintains the distance between them.

Indeed, consider vertices $u, v$ such that the shortest path $P$ between $u$ and $v$ intersects some $S_i$ (at point $b$, say). The path $P'$ between $u$ and $v$ in $T_i$ can be divided into three sections $P'_1, P'_2, P'_3$, where $P'_1$ is the shortest path from $u$ to $S_i$, $P'_3$ is the shortest path from $v$ to $S_i$, and $P'_2$ is the unique path in $S_i$ connecting the points $a$ and $c$ at which $P'_1$ and $P'_3$ meet $S_i$. (See Figure 5.1 for an illustration.) For nodes $x, y$, let $[x, y]$ denote the shortest path between $x$ and $y$ in $G$. Now since $[u, a]$ and $[v, b]$ are the shortest paths to $S_i$, $d_G(u, a) \leq d_G(u, b)$ and $d_G(v, c) \leq d_G(v, b)$. Furthermore, by the fact that $[a, c]$ is the shortest path, $d_G(a, c) \le d_G(a, u) + d_G(u, v) + d_G(v, c)$. However, the length of the path

$$\begin{aligned} d_{T_i}(u, v) &= d_G(u, a) + d_G(a, c) + d_G(c, v) \\ &\le d_G(u, a) + (d_G(a, u) + d_G(u, v) + d_G(v, c)) + d_G(c, v) \\ &\le 2(d_G(u, b) + d_G(v, b)) + d_G(u, v) = 3d_G(u, v), \end{aligned}$$

which proves the claim.

This gives us $r(n)$ trees $\{T_i\}$; it now remains to maintain distances lying within the connected components of $G - \cup_i S_i$, and hence we build tree covers for these subgraphs recursively. Since the components have size at most $\frac{2n}{3}$, we would again get at most $r(n)(\log_{3/2} n - 1)$ trees from each component; pairing them up and adding the $r(n)$ trees from the top level would give us the desired tree cover with at most $r(n) \log_{3/2} n$ trees. $\square$

A close examination of the proof of the planar separator theorem of Lipton and Tarjan [21] shows us that all planar graphs have 2-part isometric separators. Indeed, their proof first fixes an arbitrary tree $T$ in the planar graph $G$ and triangulates the graph while maintaining the planarity; it then considers cycles formed by a nontree edge $(u, v)$ and the path between $u$ and $v$ in the tree, and shows that the vertices of one such cycle form a good balanced separator.

Let us take $T$ to be the shortest-path tree from some (arbitrary) vertex $x$, and let the separator be the vertices of the fundamental cycle formed by adding the nontree $E = \{u, v\}$ to $T$. Since $T$ is a shortest-path tree rooted at $x$, there is a natural relationship between vertices in $G$, and let $\text{lca}(u, v)$ be the least common ancestor of $u$ and $v$ in $T$. The claimed 2-part isometric separator consists of the two paths from $\text{lca}(u, v)$ to $u$ and $v$.

Using this fact that planar graphs have 2-part isometric separators in conjunction with Theorem 5.1 gives us the following theorem.

THEOREM 5.2. *There exists a stretch-*3 *tree cover of size* $O(\log n)$ *for all planar graphs.*

Combining this result with Theorems 4.2 and 3.5, we immediately obtain good routing schemes for planar graphs.

COROLLARY 5.3. *There is an* $(\Delta + O(\log n \log \log n), O(\log n))$ *routing scheme for planar graphs with stretch at most* 3.

Proving such a result for broader classes of graphs, say for graphs with bounded treewidth or other graphs with small excluded minors, still remains an open problem.

**5.2. An application to small distance labelings.** In this section, we give another application of isometric separators. A *distance labeling* scheme specifies a way to label each vertex $v$ of an input graph $G$ with a label $l(v)$ drawn from a set of labels $\Sigma$, and also a uniform function $f : \Sigma \times \Sigma \to \mathbb{R}^+$ (independent of the input) which takes two labels and outputs a distance value. A *stretch-D distance labeling scheme* ensures that, given a graph $G$, the estimate given by $f$ is within a factor $D$ of the actual distance; i.e., $1 \le f(l(u), l(v))/d_G(u, v) \le D$ for all pairs of vertices $u, v \in G$. These labeling schemes have been well studied (see, e.g., [34, 26, 16]).

THEOREM 5.4. *For any planar graph* $G = (V, E)$ *with diameter* $\text{diam}(G)$*, there is a stretch-*3 *distance labeling scheme with labels of size* $O(\log n \log \text{diam}(G))$ *bits.*

The result of Theorem 5.4 should be contrasted with the result of Gavoille et al. [16], which says that planar graphs require labels of length $\Omega(n^{1/3})$ bits if no stretch is allowed. We should point out that it is possible to get a simpler $O(\log^3 n)$ bit result, by taking the $O(\log n)$ tree cover of Theorem 5.2 and using the distance labeling scheme of Peleg [26] to embed each tree with $O(\log^2 n)$ bits.

*Proof of Theorem* 5.4. For each vertex $v$, we generate $5 \log n$ coordinates $\phi_i(v)$, with each of these coordinates using $\log \operatorname{diam}(G)$ bits; these coordinates are generated in groups of five. Let us fix a hierarchical decomposition of $G$ using isometric separators.

To generate the first group for $v$, consider the 2-part isometric separator $S_0$ of $G_0 = G$. This has two shortest paths, say $P_0$ and $P_0'$, and let $a_0$ and $a_0'$ be arbitrary endpoints of $P_0$ and $P_0'$, respectively. The first two coordinates encode distances from these paths; i.e., $\phi_1(v) = d_{G_0}(v, P_0)$ and $\phi_2(v) = d_{G_0}(v, P_0')$. For the next two coordinates, let $v_0$ and $v_0'$ be the closest vertices from $v$ on $P_0$ and $P_0'$, and set $\phi_3(v) = d_{G_0}(v, a_0)$ and $\phi_4(v) = d_{G_0}(v, a_0')$. Finally, look at the graph $G \setminus S_0$, and record in the fifth coordinate the connected component in which $v$ lies, where we have numbered the components by some consistent canonical order. Set $G_1$ to be the component of $G \setminus S_0$ containing $v$, and recurse on $G_1$ to create the next set of coordinates. Note that if $v$ was in the separator, the rest of the labels would be set to $\infty$.

For the decoding function $f(u, v)$, we find the highest level of recursion $k$ in which the two vertices lie in different components (which is indicated by a difference in their values in the corresponding fifth coordinate $\phi_{5k+k}$). For each level of recursion $i$ till that level $k$, let the graph containing $u$ and $v$ be denoted by $G_i$. Now we obtain an estimate of the distance between $u$ and $v$ in $G$ by summing $d_{G_i}(v, P_i)$ and $d_{G_i}(v, P_i')$, and then adding $|d_{G_i}(v, a_i) - d_{G_i}(v, a_i')|$ to it. (All these values can be obtained from the coordinates $\phi_{5i+j}$ for $1 \leq j \leq 4$.) Finally, we take the minimum among all these estimates $0 \leq i \leq k$.

Using an argument similar to the one used in Theorem 5.2, it can be shown that if the shortest path between $u$ and $v$ intersected $S_i$, then the estimate for level $i$ would be within a factor of 3 of $d_G(u, v)$; all other estimates would be at least as large as $d_G(u, v)$. This proves the theorem. □

**6. Conclusions.** Let us conclude by mentioning some of the results that have appeared since the announcement of these results. Simultaneously and independent of our work, Thorup has used the planar separator theorem of Lipton and Tarjan in a manner similar to an approach in section 5.1 to obtain compact distance oracles for planar digraphs [32]. In fact, he uses these techniques to give a stretch-$(1 + \epsilon)$ distance labeling scheme of size $O(\log^2 n)$ for planar graphs.

Subsequently, the paper of Gupta, Kumar, and Thorup [19] has given asymptotically optimal $(\Delta + k, O(\log_k n))$ MPLS routing schemes for trees, combining these with the results of [32] to obtain $(1+\epsilon)$ stretch $(\Delta + O(\log n), O(\log n))$ MPLS routing schemes for planar graphs. Some experimental results have been given by the authors in [18], where the question of approximating the minimum label size for a given stack depth is also considered.

**Acknowledgments.** We would like to thank Jon Kleinberg, Bruce Shepherd, Éva Tardos, Peter Winkler, and Francis Zane for discussions and helpful comments.

## REFERENCES

[1] I. Althöfer, G. Das, D. Dobkin, D. Joseph, and J. Soares, *On sparse spanners of weighted graphs*, Discrete Comput. Geom., 9 (1993), pp. 81–100.

[2] M. Arias, L. J. Cowen, K. A. Laing, R. Rajaraman, and O. Taka, *Compact routing with name independence*, in Proceedings of the Fifteenth Annual ACM Symposium on Parallel Algorithms and Architectures, San Diego, CA, ACM Press, New York, 2003, pp. 184–192.

[3] D. O. Awduche, *MPLS and traffic engineering in IP networks*, IEEE Communications Magazine, 37 (1999), pp. 42–47.

[4] B. Awerbuch, A. Bar-Noy, N. Linial, and D. Peleg, *Compact distributed data structures for adaptive routing*, in Proceedings of the Twenty-First Annual ACM Symposium on Theory of Computing, Seattle, WA, ACM Press, New York, 1989, pp. 479–489.

[5] B. Awerbuch, S. Kutten, and D. Peleg, *On buffer-economical store-and-forward deadlock prevention*, in Proceedings of the Twelfth IEEE INFOCOM, San Francisco, 1991, pp. 410–414.

[6] B. Awerbuch and D. Peleg, *Routing with polynomial communication-space trade-off*, SIAM J. Discrete Math., 5 (1992), pp. 151–162.

[7] H. L. Bodlaender, *A partial k-arboretum of graphs with bounded treewidth*, Theoret. Comput. Sci., 209 (1998), pp. 1–45.

[8] B. Chandra, G. Das, G. Narasimhan, and J. Soares, *New sparseness results on graph spanners*, Internat. J. Comput. Geom. Appl., 5 (1995), pp. 125–144.

[9] *CISCO MPLS Web Page*, http://www.cisco.com/warp/public/732/Tech/mpls/.

[10] L. J. Cowen, *Compact routing with minimum stretch*, J. Algorithms, 38 (2001), pp. 170–183.

[11] B. Davie and Y. Rekhter, *MPLS: Technology and Applications*, Morgan Kaufmann (Elsevier), New York, 2000.

[12] A. Elwalid, C. Jin, S. Low, and I. Widjaja, *MATE: Multipath adaptive traffic engineering*, Computer Networks, 40 (2002), pp. 695–709.

[13] G. N. Frederickson and R. Janardan, *Designing networks with compact routing tables*, Algorithmica, 3 (1988), pp. 171–190.

[14] G. N. Frederickson and R. Janardan, *Efficient message routing in planar networks*, SIAM J. Comput., 18 (1989), pp. 843–857.

[15] C. Gavoille, *Routing in distributed networks: Overview and open problems*, Distributed Computing Column, ACM SIGACT News, 32 (2001), pp. 36–52.

[16] C. Gavoille, D. Peleg, S. Pérennes, and R. Raz, *Distance labeling in graphs*, in Proceedings of the 12th Annual ACM-SIAM Symposium on Discrete Algorithms, Washington, DC, 2001, SIAM, Philadelphia, PA, pp. 210–219.

[17] A. Gupta, A. Kumar, and R. Rastogi, *Traveling with a Pez dispenser (or, routing issues in MPLS)*, in Proceedings of the 42nd IEEE Symposium on Foundations of Computer Science (Las Vegas, NV, 2001), IEEE Computer Society, Los Alamitos, CA, 2001, pp. 148–157.

[18] A. Gupta, A. Kumar, and R. Rastogi, *Exploring the trade-off between label size and stack depth in MPLS routing*, in Proceedings of the Twenty-Second IEEE INFOCOM, San Francisco, 2003, pp. 544–554.

[19] A. Gupta, A. Kumar, and M. Thorup, *Tree based MPLS routing*, in Proceedings of the Fifteenth Annual ACM symposium on Parallel Algorithms and Architectures, San Diego, CA, ACM Press, New York, 2003, pp. 193–199.

[20] *MPLS Charter*, http://www.ietf.org/html.charters/mpls-charter.html.

[21] R. J. Lipton and R. E. Tarjan, *A separator theorem for planar graphs*, SIAM J. Appl. Math., 36 (1979), pp. 177–189.

[22] G. A. Margulis, *Explicit group-theoretic constructions of combinatorial schemes and their applications in the construction of expanders and concentrators*, Problems in Information Transmission, 24 (1988), pp. 51–60.

[23] J. Matoušek, *On embedding trees into uniformly convex Banach spaces*, Israel J. Math., 114 (1999), pp. 221–237; Czech version in *Lipschitz Distance of Metric Spaces*, C.Sc. degree thesis, Charles University, 1990.

[24] D. Mitra and K. G. Ramakrishnan, *Engineering of multiservice, multipriority networks*, Bell Labs Technical J., 6 (2001), pp. 139–151.

[25] *Nortel MPLS Web Page*, http://www.nortelnetworks.com/corporate/technology/mpls/index.html.

[26] D. Peleg, *Proximity-preserving labeling schemes and their applications*, in Proceedings of the 25th Workshop on Graph-Theoretic Concepts in Computer Science, Lecture Notes in Comput. Sci. 1665, Springer-Verlag, New York, 1999, pp. 30–41.

[27] D. Peleg, *Distributed Computing: A Locality-Sensitive Approach*, SIAM Monogr. Discrete Math. Appl. 5, SIAM, Philadelphia, PA, 2000.

[28] D. Peleg and E. Upfal, *A trade-off between space and efficiency for routing tables*, J. ACM, 36 (1989), pp. 510–530.

[29] D. Peleg and E. Upfal, *A trade-off between space and efficiency for routing tables*, in Proceedings of the 20th Annual ACM Symposium on Theory of Computing, Dallas, TX, 1998, ACM Press, New York, pp. 43–52.

[30] E. C. Rosen, D. Tappan, Y. Rekhter, G. Federkow, D. Farinacci, T. Li, and A. Conta, *MPLS Label Stack Encoding (RFC* 3032*)*, http://www.ietf.org/rfc/rfc3032.txt, 2001.

[31] E. C. Rosen, A. Viswanathan, and R. Callon, *MultiProtocol Label Switching Architecture (RFC* 3031*)*, http://www.ietf.org/rfc/rfc3031.txt, 2001.
[32] M. Thorup, *Compact oracles for reachability and approximate distances in planar digraphs*, in Proceedings of the 42nd Annual IEEE Symposium on Foundations of Computer Science, Las Vegas, NV, 2001, pp. 242–251.
[33] M. Thorup and U. Zwick, *Approximate distance oracles*, in Proceedings of the 33rd Annual ACM Symposium on Theory of Computing, Crete, Greece, 2001, ACM Press, New York, pp. 183–192.
[34] P. Winkler, *Proof of the squashed cube conjecture*, Combinatorica, 3 (1983), pp. 135–139.

SIAM J. COMPUT.
Vol. 34, No. 2, pp. 475–497

# THE SIMPLEX ALGORITHM IN DIMENSION THREE*

VOLKER KAIBEL[†], RAFAEL MECHTEL[‡], MICHA SHARIR[§], AND GÜNTER M. ZIEGLER[‡]

**Abstract.** We investigate the worst-case behavior of the simplex algorithm on linear programs with three variables, that is, on 3-dimensional simple polytopes. Among the pivot rules that we consider, the "random edge" rule yields the best asymptotic behavior as well as the most complicated analysis. All other rules turn out to be much easier to study, but also produce worse results: Most of them show essentially worst-possible behavior; this includes both Kalai's "random-facet" rule, which without dimension restriction is known to be subexponential, and Zadeh's deterministic history-dependent rule, for which no nonpolynomial instances in general dimensions have been found so far.

**Key words.** linear programming, random edge, pivot rule, linearity coefficient

**AMS subject classifications.** 65K05, 90C05, 90C35

**DOI.** 10.1137/S0097539703434978

## 1. Introduction.

The simplex algorithm is a fascinating method for at least three reasons: For computational purposes it is still the most efficient general tool for solving linear programs; from a complexity point of view it is the most promising candidate for a strongly polynomial time linear programming algorithm; and last but not least, geometers are pleased by its inherent use of the structure of convex polytopes.

The essence of the method can be described geometrically: Given a convex polytope $P$ by means of inequalities, a linear functional $\varphi$ "in general position," and some vertex $v_{\text{start}}$, the simplex algorithm chooses an edge to a neighboring vertex along which $\varphi$ decreases strictly. Iterating this yields a $\varphi$-monotone edge-path. Such a path can never get stuck, and will end at the unique $\varphi$-minimal ("optimal") vertex of $P$.

Besides implementational challenges, a crucial question with respect to efficiency asks for a suitable *pivot rule* that prescribes how to proceed with the monotone path at any vertex. Since Dantzig invented the simplex algorithm in the late 1940's [4], a great variety of pivot rules have been proposed. Most of them (including Dantzig's original "largest coefficient rule") have subsequently been shown to lead to exponentially long paths in the worst case. (See [1] for a survey.) Prominent exceptions are Zadeh's

*Received by the editors September 22, 2003; accepted for publication (in revised form) July 29, 2004; published electronically February 3, 2005. The first and second authors were supported by the DFG-Forschergruppe *Algorithmen, Struktur, Zufall* (FOR 413/1-1, Zi 475/3-1). Part of this work was done during the workshop "Towards the Peak" at La Claustra, Switzerland, in August, 2001.

http://www.siam.org/journals/sicomp/34-2/43497.html

[†]DFG Research Center *MATHEON*, MA 6–2, TU Berlin, 10623 Berlin, Germany (kaibel@math.tu-berlin.de).

[‡]MA 6–2, TU Berlin, 10623 Berlin, Germany (mechtel@math.tu-berlin.de, ziegler@math.tu-berlin.de). The fourth author acknowledges partial support by the Deutsche Forschungs-Gemeinschaft (DFG), FZT86, ZI 475/3, and ZI 475/4 and by the GIF project *Combinatorics of Polytopes in Euclidean Spaces* (I-624-35.6/1999).

[§]School of Computer Science, Tel Aviv University, Tel-Aviv 69978, Israel, and Courant Institute of Mathematical Sciences, New York University, New York, NY 10012 (michas@post.tau.ac.il). The work of this author was supported by NSF grants CCR-97-32101 and CCR-00-98246, by a grant from the U.S.-Israeli Binational Science Foundation, by a grant from the Israel Science Fund (for a Center of Excellence in Geometric Computing), and by the Hermann Minkowski–MINERVA Center for Geometry at Tel Aviv University.

history-dependent "least entered" rule and several randomized pivot rules. Particularly remarkable is the "random facet" rule proposed by Kalai [9]; its expected path length for all instances is bounded subexponentially in the number of facets. See also Matoušek, Sharir, and Welzl [14].

In this paper, we analyze the worst-case behavior of the simplex method on 3-dimensional simple polytopes for some well-known pivot rules. At first glance, the 3-dimensional case may seem trivial, since by Euler's formula a 3-polytope with $n$ facets has at most $2n-4$ vertices (with equality if and only if the polytope is simple), and there are examples where $n-3$ steps are needed for any monotone path to the optimum (see, e.g., Figure 2.1). Therefore, for *any* pivot rule the simplex algorithm is linear, with at least $n-3$ and at most $2n-5$ steps in the worst case. However, no pivot rule is known that would work with at most $n-3$ steps.

In order to summarize our results, we define the following measure of quality. Fix a pivot rule $\mathcal{R}$. For every 3-dimensional polytope $P \subset \mathbb{R}^3$ and for every linear functional $\varphi : \mathbb{R}^3 \longrightarrow \mathbb{R}$ *in general position* with respect to $P$ (i.e., no two vertices of $P$ have the same $\varphi$-value), denote by $\lambda_{\mathcal{R}}(P, v_{\text{start}})$ the path length (expected path length, if $\mathcal{R}$ is randomized) produced by the simplex algorithm with the pivot rule $\mathcal{R}$, when started at vertex $v_{\text{start}}$. The *linearity coefficient* of $\mathcal{R}$ is

$$\Lambda(\mathcal{R}) \; := \; \limsup_{n(P)\to\infty} \left\{ \frac{\lambda_{\mathcal{R}}(P, v_{\text{start}})}{n(P)} \; : \; P, \varphi, v_{\text{start}} \text{ as above} \right\},$$

where $n(P)$ is the number of facets of $P$. With the usual simplifications for a geometric analysis (cf. [13], [20, Lecture 3], [1]), we may restrict our attention to *simple* 3-dimensional polytopes $P$ (where each vertex is contained in precisely three facets). Thus we consider only 3-dimensional polytopes $P$, with $n = n(P)$ facets, $3n-6$ edges, and $2n-4$ vertices. By the discussion above, the linearity coefficient satisfies $1 \le \Lambda(\mathcal{R}) \le 2$ for every pivot rule $\mathcal{R}$.

The most remarkable aspect of the picture that we obtain, in section 3, is that the "random edge" rule ("RE" for short) performs quite well (as it is conjectured for general dimensions), but it is quite tedious to analyze (as has already been observed for general dimensions). The following bounds for the random edge rule,

$$1.3473 \quad \le \quad \Lambda(\text{RE}) \quad \le \quad 1.4943,$$

are our main results. Thus we manage to separate $\Lambda(\text{RE})$ from the rather easily achieved lower bound of $\frac{4}{3}$, as well as from the already nontrivial upper bound of $\frac{3}{2}$.

On the other hand, in section 4 we prove that the linearity coefficient for the "greatest decrease" pivot rule is $\Lambda(\text{GD}) = \frac{3}{2}$, while many other well-known rules have linearity coefficient $\Lambda = 2$, including the largest coefficient, least index, steepest decrease, and the shadow vertex rules, as well as Zadeh's history-dependent least entered rule (not known to be superpolynomial in general), and Kalai's random facet rule (known to be subexponential in general).

## 2. Basics.

Klee [12] proved in 1965 that the "monotone Hirsch conjecture" is true for 3-dimensional polytopes; that is, whenever the graph of a 3-dimensional polytope $P$ with $n$ facets is oriented by means of a linear functional in general position, there is a monotone path of length at most $n-3$ from any vertex to the sink $v_{\min}$. (See Klee and Kleinschmidt [13] for a survey of the Hirsch conjecture and its ramifications.) Unfortunately, Klee's proof is not based on a pivot rule.

THEOREM 2.1 (Klee [12]). *For any simple 3-polytope $P \subset \mathbb{R}^3$, a linear functional $\varphi : \mathbb{R}^3 \longrightarrow \mathbb{R}$ in general position for $P$, and any vertex $v_{start}$ of $P$, there is a $\varphi$-monotone path from $v_{start}$ to the $\varphi$-minimal vertex $v_{\min}$ of $P$ that does not revisit any facet.*

*In particular, there is a $\varphi$-monotone path from $v_{start}$ to $v_{\min}$ of length at most $n - 3$.*

It is not too hard to come up with examples showing that the bound provided by Theorem 2.1 is best possible. One of the constructions will be important for our treatment later on, so we describe it below in Figure 2.1.

A particularly useful tool for constructing LP-oriented 3-polytopes is the following result due to Mihalisin and Klee. It is stated in a slightly weaker version in their paper, but their proof actually shows the following.

THEOREM 2.2 (Mihalisin and Klee [17]). *Let $G = (V, E)$ be a planar 3-connected graph, $f : V \longrightarrow \mathbb{R}$ any injective function, and denote by $\vec{G}$ the acyclic oriented graph obtained from $G$ by directing each edge to its endnode with the smaller $f$-value. Then the following are equivalent:*

1. *There exist a polytope $P \subset \mathbb{R}^3$ and a linear functional $\varphi : \mathbb{R}^3 \longrightarrow \mathbb{R}$ in general position for $P$ such that $G$ is isomorphic to the graph of $P$ and, for every $v \in V$, $f(v)$ agrees with the $\varphi$-value of the vertex of $P$ corresponding to $v$.*
2. *Both* (a) *and* (b) *hold:*
   (a) *$\vec{G}$ has a unique sink in every facet (induced nonseparating cycle) of $G$, and*
   (b) *there are three node-disjoint monotone paths joining the (unique) source to the (unique) sink of $\vec{G}$.*

Here the fact that the source and the sink of $\vec{G}$ are unique (referred to in condition (b)) follows from (a); cf. Joswig, Kaibel, and Körner [8]. Equipped with Theorem 2.2, one readily verifies that the family of directed graphs indicated in Figure 2.1 ($n \geq 4$) can be realized as convex 3-polytopes, with associated linear functionals, demonstrating that Klee's bound of $n - 3$ on the length of a shortest monotone path cannot be improved.

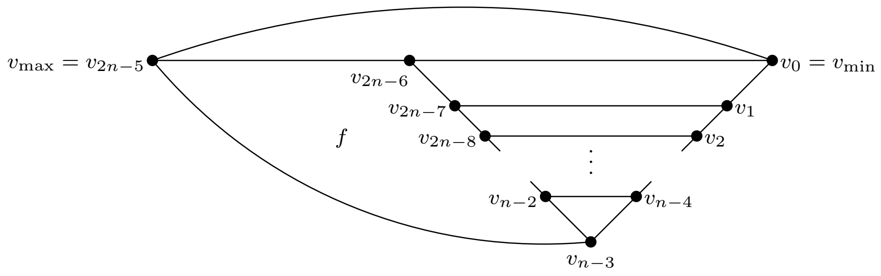


FIG. 2.1. *A worst case example for Klee's theorem, starting at $v_{n-3}$ (and for Bland's rule, starting at $v_{2n-6}$; see section 4.1). All edges are oriented from left to right.*

**3. The random edge rule.** At any nonoptimal vertex, the *random edge* pivot rule takes a step to one of its improving neighbors, chosen uniformly at random. Thus the expected number $\mathbf{E}(v)$ of steps that the random edge rule would take from a given vertex $v$ to the optimal one $v_{\min}$ may be computed recursively as

$$\mathbf{E}(v) \;=\; 1 + \frac{1}{|\,\delta^{\text{out}}(v)|} \sum_{u:(v,u)\in\delta^{\text{out}}(v)} \mathbf{E}(u), \tag{3.1}$$

where $\delta^{\text{out}}(v)$ denotes the set of edges that leave $v$ (that is, lead to *better* vertices), so that $|\,\delta^{\text{out}}(v)|$ is the number of neighbors of $v$ whose $\varphi$-value is smaller than that of $v$.

Despite its simplicity and its (deceptively) simple recursion, this rule has by now resisted several attempts to analyze its worst-case behavior, with a few exceptions for special cases, namely linear assignment problems (Tovey [19]), the Klee–Minty cubes (Kelly [11] and Gärtner, Henk, and Ziegler [6]), and $d$-dimensional linear programs with at most $d+2$ inequalities (Gärtner et al. [7]). All known results leave open the possibility that the expected number of steps taken by the random edge rule on a $d$-dimensional linear program with $n$ inequalities could be bounded by a polynomial, perhaps even by $O(n^2)$ or $O(dn)$, where $n$ is the number of facets.

However, Matoušek and Szabó [15] recently showed that the random edge rule does not have a polynomially bounded running time on the larger class of *acyclic unique sink orientations* (*AUSO*'s), i.e., acyclic orientations of the graph of a polytope that induce unique sinks in all nonempty faces (cf. condition 2(a) in Theorem 2.2). They exhibited particular AUSO's on $d$-dimensional cubes for which random edge needs at least $\text{const}\cdot 2^{\text{const}\cdot d^{1/3}}$ steps.

**3.1. Lower bounds.** The lower bound calculations appear to be much simpler if we do not use the recursion given above but instead use a "flow model." For this, fix a starting vertex $v_{\text{start}}$, and denote by $p(v)$ the probability that the vertex $v$ will be visited by a random edge path from $v_{\text{start}}$ to $v_{\min}$, and similarly by $p(e)$ the probability that a directed edge $e$ will be traversed. Then the probability that a vertex $v$ is visited is the sum of the probabilities that the edges leading into $v$ are traversed,

$$p(v) = \sum_{e\in\delta^{\text{in}}(v)} p(e),$$

if $v$ is not the starting vertex. (Here $\delta^{\text{in}}(v)$ denotes the set of edges that enter $v$.) Furthermore, by definition of the random edge rule we have

$$p(e) \;=\; \frac{1}{|\,\delta^{\text{out}}(v)|}p(v) \qquad \text{for all } e\in\delta^{\text{out}}(v) \tag{3.2}$$

at each nonoptimal vertex. The random edge rule thus induces a flow $(p(e))_{e\in E}$ of value 1 from $v_{\text{start}}$ to $v_{\min}$. The expected path length $\mathbf{E}(v_{\text{start}})$ is then given by

$$\mathbf{E}(v_{\text{start}}) \;=\; \sum_{e\in E} p(e), \tag{3.3}$$

and we refer to it as the *cost* of the flow $(p(e))_{e\in E}$.

THEOREM 3.1. *The linearity coefficient of the random edge rule satisfies*

$$\Lambda(\text{RE}) \;\geq\; \frac{1897}{1408} \;>\; 1.3473.$$

*Proof.* We describe a family of linear programs (LPs) which show the above lower bound on the linearity coefficient. We start with the graph of the dual-cyclic polytope

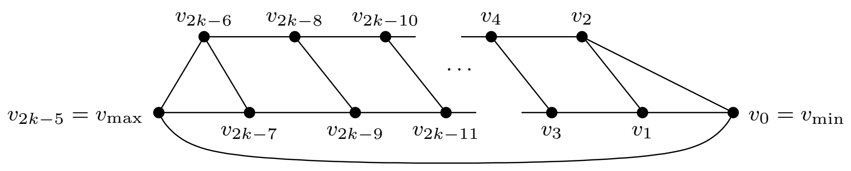


FIG. 3.1. *Lower bound construction for the random edge rule: the backbone polytope. All edges are oriented from left to right.*

$C_3(k)^\Delta$ with the orientation depicted in Figure 3.1, and refer to this as the *backbone* of the construction.

Starting at the vertex $v_{2k-7}$, the simplex algorithm will take the path along the $k-2$ vertices $v_{2k-7}, v_{2k-9}, \ldots, v_3, v_1, v_0$. Replacing each vertex in the path by a copy of the digraph depicted in Figure 3.2—called a *configuration* in the following—yields the desired LP. The corresponding feasible polytope can be constructed explicitly by applying ten suitable successive vertex cuts at each vertex $v_i$ of the backbone. Alternatively, one can check that the orientations we get satisfy the conditions of Theorem 2.2.

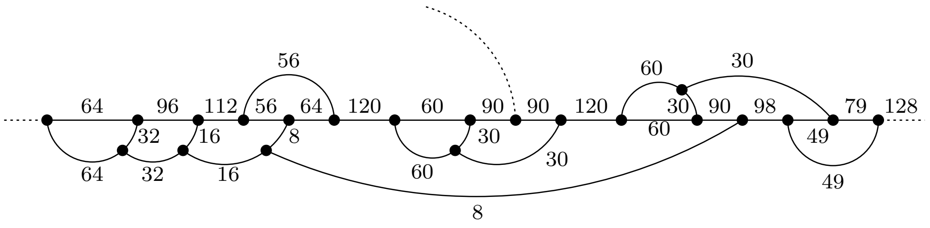


FIG. 3.2. *Lower bound construction for the random edge rule: the configuration. All edges are oriented from left to right. The target of the rightmost edge (not shown) is the starting node of the next configuration. The middle dotted edge enters the configuration from the corresponding vertex of the top backbone row. The actual flow at each edge is* 1/128 *times the number written next to the edge.*

The maximal and minimal vertex of each configuration is visited with probability 1. We send 128 units of flow (each of value $\frac{1}{128}$) through each configuration according to (3.2); see Figure 3.2. This yields the flow-cost of $\frac{1897}{128}$ for each of the $k-2$ configurations. (The last configuration produces a flow-cost of $\frac{1897}{128} - 1$ only, as it does not have a leaving edge.)

We take the maximal vertex of the configuration at $v_{2k-7}$ as the starting vertex $v_{\text{start}}$. Using (3.3), we obtain for the expected cost $\mathbf{E}(v_{\text{start}})$

$$\mathbf{E}(v_{\text{start}}) = (k-2)\frac{1897}{128} - 1.$$

With $n = k + 10(k-2)$, this yields

$$\mathbf{E}(v_{\text{start}}) \;=\; \frac{n-2}{11} \cdot \frac{1897}{128} - 1 \;=\; \frac{1897}{1408}n - \frac{5202}{1408},$$

which proves the lower bound. $\square$

The configuration depicted in Figure 3.2 was found by complete enumeration of the acyclic orientations satisfying condition (a) of Theorem 2.2 (AUSOs) on 3-polytopes with $n \leq 12$ facets. In particular, our proof of Theorem 3.1 includes a worst-possible example for $n = 12$. We refer to Mechtel [16] for more details of the search procedure, as well as for a detailed analysis of the properties of worst-case examples for the random edge rule.

**3.2. Upper bounds.**

THEOREM 3.2. *The linearity coefficient of the random edge rule satisfies*

$$\Lambda(\mathrm{RE}) \ \le\ \frac{130}{87} < 1.4943.$$

*Proof.* Consider any linear program on a simple 3-polytope with $n$ facets, with a linear objective function $\varphi$ in general position. We will refer to the $\varphi$-value of a vertex as its "height." A 1-*vertex* will denote a vertex with exactly one neighbor that is lower with respect to $\varphi$. Similarly, a 2-*vertex* has exactly two lower neighbors. Consequently, from any 1-vertex the random edge rule proceeds deterministically to the unique improving neighbor, and from any 2-vertex it proceeds to one of the two improving neighbors, each with probability $\frac{1}{2}$.

Basic counting yields that our LP has exactly $(n-3)$ 1-vertices and $(n-3)$ 2-vertices in addition to the unique maximal vertex $v_{\max}$ and the unique minimal vertex $v_{\min}$, which have 3 and 0 lower neighbors, respectively. For the following, we also assume that the vertices are sorted and labelled $v_{2n-5}, \dots, v_1, v_0$ in decreasing order of their objective function values, with $v_{\max} = v_{2n-5}$ and $v_{\min} = v_0$.

For any vertex $v$, let $N_1(v)$ (resp., $N_2(v)$) denote the number of 1-vertices (resp., 2-vertices) that are not higher than $v$ (including $v$ itself). Set $N(v) = N_1(v) + N_2(v)$. For all vertices $v$ other than the maximal one, this is the number of vertices lower than $v$; that is, $N(v_i) = i$ for all $i \neq 2n-5$.

We will establish the following generic inequality for all $v \neq v_{\max}$ by induction on $N(v)$:

$$\mathbf{E}(v) \ \le\ \alpha N_1(v) + \beta N(v) + \gamma. \tag{3.4}$$

Here, $\alpha$ and $\beta$ are constants whose values will be fixed later in the induction step, and $\gamma$ is a constant whose value will be fixed in the induction base case. The inductive step will be subdivided into 24 distinct cases. Each case depends on a linear inequality on $\alpha$ and $\beta$ that, when satisfied, justifies the induction step in that case. Since our case analysis is complete, we have a proof of (3.4) for any pair $(\alpha, \beta)$ that satisfies all the 24 inequalities.

Because we always have $N_1(v), N_2(v) \le n-3$, we obtain

$$\begin{aligned}\mathbf{E}(v) &\le\ \alpha N_1(v) + \beta(N_1(v) + N_2(v)) + \gamma = (\alpha+\beta)N_1(v) + \beta N_2(v) + \gamma \\ &\le\ (\alpha + 2\beta)(n-3) + \gamma\end{aligned}$$

for $v \neq v_{\max}$. The single vertex $v_{\max}$ is irrelevant for the asymptotic considerations. Thus we minimize $\alpha + 2\beta$ subject to the linear constraints posed by the various cases; this leads to an LP in two variables with 24 constraints, whose optimal solution is $(\alpha, \beta) = (\frac{46}{87}, \frac{42}{87})$, of value $\frac{130}{87} < 1.4943$. This yields the upper bound on $\Lambda(\mathrm{RE})$ stated in the theorem.

Before starting the inductive proof, we modify the original polytope to a new one $P'$ (with $v'_{\min}$ being the new sink) by cutting off the sink of the given simple 3-polytope ten times in succession, in the following way: (1) For all original vertices $v \neq v_{\min}$, the expected number of vertices on a path to $v'_{\min}$ is larger than the expected number of vertices on a path to $v_{\min}$ in $P$; (2) every original vertex lies at distance at least five from $v'_{\min}$; (3) the newly generated 20 vertices, together with the sink, are the 21 lowest vertices of $P'$.

By choosing the constant $\gamma$ sufficiently large, we ensure that $\mathbf{E}(v) \le \gamma$ holds for all vertices $v$ of $P'$ with $N(v) \le 20$.

We prove Theorem 3.2 for the new polytope $P'$. Since the expected path lengths only increase, this also implies the theorem for the original polytope. The preceding analysis implies that (3.4) holds for all $v$ satisfying $N(v) \le 20$, which establishes the base case of the induction.

Suppose now that (3.4) holds for all vertices lower than some vertex $v$ with $N(v) > 20$. By an appropriate unwinding of the recursion (3.1), we express $\mathbf{E}(v)$ in terms of the expected cost $\mathbf{E}(w_i)$ of certain vertices $w_i$ that are reachable from $v$ via a few downward edges. The general form of such a recursive expression will be

$$\mathbf{E}(v) \;=\; c + \sum_{i=1}^{k} \lambda_i \mathbf{E}(w_i),$$

where $\lambda_i > 0$ for $i = 1, \dots, k$, and $\sum_i \lambda_i = 1$.

Since we assume by induction that $\mathbf{E}(w_i) \le \alpha N_1(w_i) + \beta N(w_i) + \gamma$, for each $i$, it suffices to show that

$$\sum_{i=1}^{k} \alpha\lambda_i \left(N_1(v) - N_1(w_i)\right) + \sum_{i=1}^{k} \beta\lambda_i \left(N(v) - N(w_i)\right) \;\ge\; c.$$

In our analysis, we will need to classify some of the vertices $w_i$ as being either 1-vertices or 2-vertices, for which we need to make sure that they are not the sink. By inspecting all 24 cases, we see that this will be the case if $v$ lies at distance at least five from $v'_{\min}$. Since we assume that $N(v) > 20$, the vertex $v$ must be an original one; thus it has distance at least five from $v'_{\min}$.

Write

$$\Delta_1(w_i) := N_1(v) - N_1(w_i), \qquad \Delta(w_i) := N(v) - N(w_i),$$

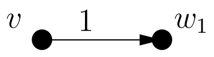
for $i = 1, \dots, k$. (These terms are defined with respect to the vertex $v$ that is currently considered.) Here $\Delta(w_i)$ is the *distance* between $v$ and $w_i$, that is, one plus the number of vertices between $v$ and $w_i$ in the numbering of the vertices $(v_{2n-5}, \dots, v_0)$ detailed above. Clearly $\Delta(w_i) \ge \Delta_1(w_i)$.

We thus need to show that for each vertex $v$,

$$\alpha \sum_{i=1}^{k} \lambda_i \Delta_1(w_i) + \beta \sum_{i=1}^{k} \lambda_i \Delta(w_i) \ge c. \tag{3.5}$$

At this point we start our case analysis.

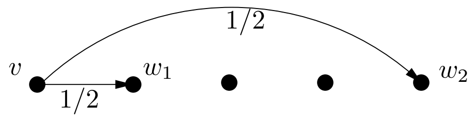
*Case* 1: $v$ is a 1-vertex. Let $w_1$ denote the target of the unique downward edge emanating from $v$, as in the following figure, where (here and in all subsequent figures) each edge is labelled by the probability of reaching it from $v$.



In this case, $\mathbf{E}(v) = 1 + \mathbf{E}(w_1)$. In the setup presented above, we have $\lambda_1 = 1$, $c = 1$, $\Delta_1(w_1) \ge 1$, and $\Delta(w_1) \ge 1$; thus (3.5) is implied by

$$\alpha + \beta \ge 1. \tag{3.6}$$

*Case* 2: $v$ is a 2-vertex. Let $w_1$ and $w_2$ denote the targets of the two downward edges emanating from $v$, where $w_2$ is lower than $w_1$.



We have

$$\mathbf{E}(v) \ = \ 1 + \frac{1}{2}\mathbf{E}(w_1) + \frac{1}{2}\mathbf{E}(w_2),$$

and hence we need to require that

$$\frac{\alpha}{2}\Delta_1(w_1) + \frac{\alpha}{2}\Delta_1(w_2) + \frac{\beta}{2}\Delta(w_1) + \frac{\beta}{2}\Delta(w_2) \ \geq \ 1.$$

Note that $\Delta(w_2) > \Delta(w_1) \geq 1$.

*Case* 2.a: $\Delta(w_2) \geq 4$ (as in the preceding figure). Ignoring the effect of the $\Delta_1(w_j)$'s, it suffices to require that

$$\frac{\beta}{2}\Delta(w_1) + \frac{\beta}{2}\Delta(w_2) \geq 1,$$

which will follow if

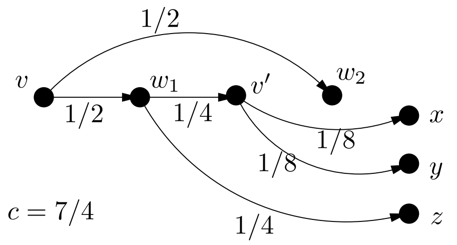
$$\beta \geq \frac{2}{5}. \tag{3.7}$$

*Case* 2.b.i: $\Delta(w_2) = 3$ and one of the two vertices above $w_2$ and below $v$ is a 1-vertex. In this case $\Delta_1(w_2) \geq 1$ and $\Delta(w_1) + \Delta(w_2) \geq 4$, so (3.5) is implied by

$$\frac{1}{2}\alpha + 2\beta \geq 1. \tag{3.8}$$

*Case* 2.b.ii: $\Delta(w_2) = 3$ and the two vertices between $v$ and $w_2$ are 2-vertices. Denote the second intermediate vertex as $v'$. We may assume that $v'$ is reachable from $v$ (that is, from $w_1$); otherwise we can ignore it and reduce the situation to Case 2.c treated below (by choosing another ordering of the vertices producing the same oriented graph). Three subcases can arise.

First, assume that none of the three edges that emanate from $w_1$ and $v'$ further down reaches $w_2$. Denote by $x, y$ the two downward neighbors of $v'$, and by $z$ the downward neighbor of $w_1$ other than $v'$. The vertices $x, y, z$ need not be distinct (except that $x \neq y$), but none of them coincides with $w_2$.



We have here $c = 7/4$.

To make the analysis simpler to follow visually, we present it in a table. Each row denotes one of the target vertices $w_2, x, y, z$, "multiplied" by the probability of reaching it from $v$. The left (resp., right) column denotes a lower bound on the corresponding quantities $\Delta_1(\cdot)$ (resp., $\Delta(\cdot)$). To obtain an inequality that implies (3.5), one has to multiply each entry in the left (resp., right) column by the row probability times $\alpha$ (resp., times $\beta$), and require that the sum of all these terms be $\geq c$.

| | $\alpha\Delta_1$ | $\beta\Delta$ |
|---|---|---|
| $1/2w_2$ | 0 | 3 |
| $1/8x$ | 0 | 4 |
| $1/8y$ | 0 | 5 |
| $1/4z$ | 0 | 4 |

Note the following: (a) We do not assume that the rows represent distinct vertices (in fact, $x = z$ is implicit in the table); this does not cause any problem in applying the rule for deriving an inequality from the table. (b) We have to squeeze the vertices so as to make the resulting inequality as sharp (and difficult to satisfy) as possible; thus we made one of $x, y$ the farthest vertex, because making $z$ the farthest vertex would have made the inequality easier to satisfy.

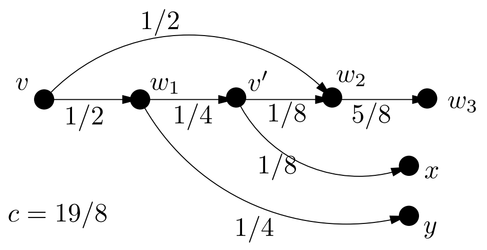
We thus obtain

$$\left(\frac{3}{2} + \frac{4}{8} + \frac{5}{8} + \frac{4}{4}\right)\beta \geq \frac{7}{4},$$

or

$$\beta \geq \frac{14}{29}. \tag{3.9}$$

Next, assume that $w_2$ is connected to $v'$. In this case $w_2$ is a 1-vertex, and we extend the configuration to include its unique downward neighbor $w_3$.



Let $x$ denote the other downward neighbor of $v'$, and let $y$ denote the other downward neighbor of $w_1$. In the following table, the "worst" case is to make $w_3$ and $y$ coincide, and make $x$ the farthest vertex.

| | $\alpha\Delta_1$ | $\beta\Delta$ |
|---|---|---|
| $5/8w_3$ | 1 | 4 |
| $1/8x$ | 1 | 5 |
| $1/4y$ | 1 | 4 |

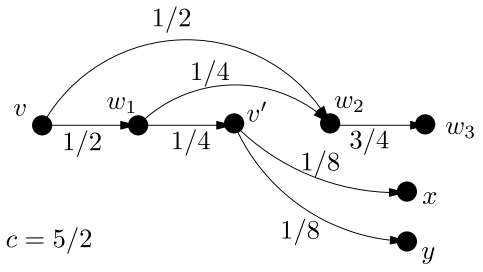
We then obtain

$$\alpha + \left(\frac{20}{8} + \frac{5}{8} + \frac{4}{4}\right)\beta \geq \frac{19}{8},$$

or

$$\alpha + \frac{33}{8}\beta \geq \frac{19}{8}. \tag{3.10}$$

Finally, assume that $w_2$ is connected to $w_1$. Here too $w_2$ is a 1-vertex, and we extend the configuration to include its unique downward neighbor $w_3$.



Denoting by $x, y$ the two downward neighbors of $v'$, our table and resulting inequality become

(3.11)

| | $\alpha\Delta_1$ | $\beta\Delta$ |
|---|---|---|
| $3/4w_3$ | 1 | 4 |
| $1/8x$ | 1 | 4 |
| $1/8y$ | 1 | 5 |

$$\alpha + \frac{33}{8}\beta \geq \frac{5}{2},$$

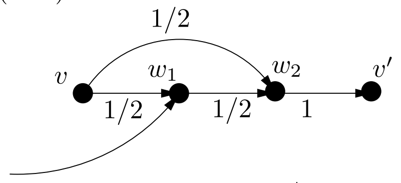
which, by the way, is stronger than (3.10).

*Case* 2.c: $\Delta(w_2) = 2$. Hence, the only remaining case is that $w_1$ and $w_2$ are the two vertices immediately following $v$.

*Case* 2.c.i: $w_1$ is a 1-vertex (whose other upward neighbor lies above $v$). Its unique downward edge ends at some vertex which is either $w_2$ or lies below $w_2$.

Assume first that this vertex coincides with $w_2$, which makes $w_2$ a 1-vertex, whose unique downward neighbor is denoted as $v'$. The local structure, table, and inequality are

(3.12)



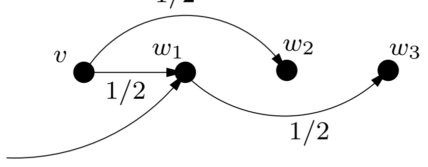
$c = 5/2$

| | $\alpha\Delta_1$ | $\beta\Delta$ |
|---|---|---|
| $v'$ | 2 | 3 |

$$2\alpha + 3\beta \geq \frac{5}{2}.$$

Suppose next that the downward neighbor $w_3$ of $w_1$ lies below $w_2$. We get

(3.13)

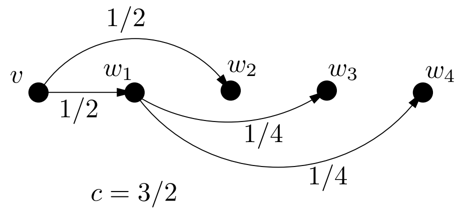


$c = 3/2$

| | $\alpha\Delta_1$ | $\beta\Delta$ |
|---|---|---|
| $1/2w_2$ | 1 | 2 |
| $1/2w_3$ | 1 | 3 |

$$\alpha + \frac{5}{2}\beta \geq \frac{3}{2}.$$

*Case* 2.c.ii: $w_1$ is a 2-vertex, both of whose downward neighbors lie strictly below $w_2$. Denote these neighbors as $w_3, w_4$, with $w_3$ lying above $w_4$.



We may assume that $\Delta(w_3) = 3$ (i.e., there is no vertex between $w_2$ and $w_3$), since the case $\Delta(w_3) \geq 4$ is already covered by (3.9).

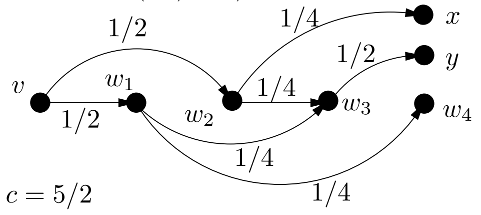
*Case* 2.c.ii.1: $w_2$ is a 1-vertex. Then the table and inequality become

(3.14)

| | $\alpha\Delta_1$ | $\beta\Delta$ |
|---|---|---|
| $1/2w_2$ | 0 | 2 |
| $1/4w_3$ | 1 | 3 |
| $1/4w_4$ | 1 | 4 |

$$\frac{1}{2}\alpha + \frac{11}{4}\beta \geq \frac{3}{2}.$$

*Case* 2.c.ii.2: $w_2$ is a 2-vertex but $w_3$ is a 1-vertex. Then $w_3$ (which satisfies $\Delta(w_3) = 3$) is connected either to $w_2$ or to a vertex above $v$. In the former case, let $x$ denote the other downward neighbor of $w_2$, and let $y$ denote the unique downward neighbor of $w_3$. The local structure looks like this (with $x, y, w_4$ not necessarily distinct but all below $w_3$ due to $\Delta(w_3) = 3$):



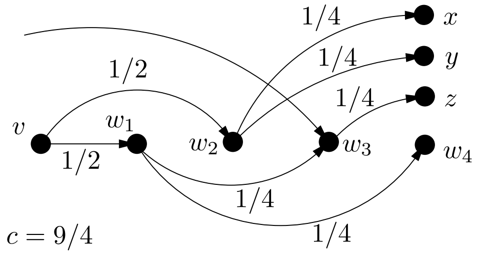
The (worst) table and inequality are

(3.15)

| | $\alpha\Delta_1$ | $\beta\Delta$ |
|---|---|---|
| $1/4x$ | 1 | 4 |
| $1/2y$ | 1 | 4 |
| $1/4w_4$ | 1 | 5 |

$$\alpha + \frac{17}{4}\beta \geq \frac{5}{2}.$$

The next case is where the other upward neighbor of $w_3$ lies above $v$. Let $x, y$ denote the two downward neighbors of $w_2$, and let $z$ denote the unique downward neighbor of $w_3$. (Again, $x, y, z, w_4$ need not be distinct, but $x \neq y$ and they are all below $w_3$ due to $\Delta(w_3) = 3$.) The local structure is



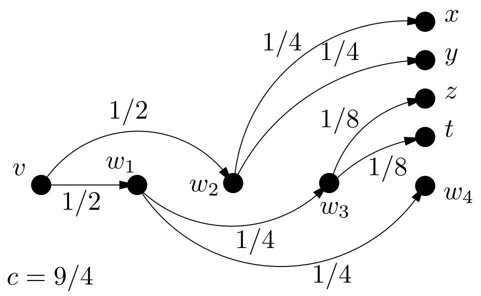
The (worst) table and inequality become

(3.16)

| | $\alpha\Delta_1$ | $\beta\Delta$ |
|---|---|---|
| $1/4x$ | 1 | 4 |
| $1/4y$ | 1 | 5 |
| $1/4z$ | 1 | 4 |
| $1/4w_4$ | 1 | 5 |

$$\alpha + \frac{9}{2}\beta \geq \frac{9}{4}.$$

*Case* 2.c.ii.3: Both $w_2$ and $w_3$ are 2-vertices. We have to consider the following type of configuration (where $x, y, z, t, w_4$ need not all be distinct, but $x \neq y$ and $z \neq t$, and we may assume $x \neq t$, $y \neq z$; also, because $\Delta(w_3) = 3$, both $x$ and $y$ are lower than $w_3$):



Intuitively, a worst table is obtained by "squeezing" $x$, $y$, $z$, $t$, and $w_4$ as much to the left as possible, placing two of them at distance 4 from $v$, two at distance 5, and one at distance 6. However, squeezing them this way will make some pairs of them coincide and form 1-vertices, which will affect the resulting tables and inequalities.

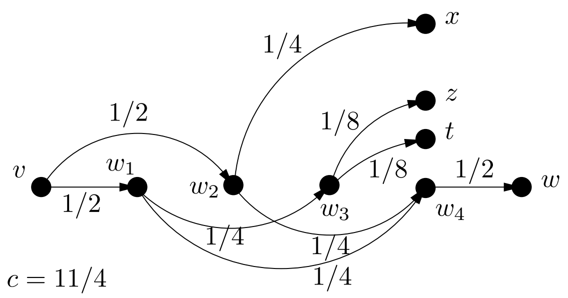
Suppose first that among the three "heavier" targets $x, y, w_4$, at most one lies at distance 4 from $v$. The worst table and the associated inequality are (recall that $x \neq y$):

(3.17)

| | $\alpha\Delta_1$ | $\beta\Delta$ |
|---|---|---|
| $1/4x$ | 0 | 4 |
| $1/4y$ | 0 | 5 |
| $1/8z$ | 0 | 4 |
| $1/8t$ | 0 | 6 |
| $1/4w_4$ | 0 | 5 |

$$\frac{19}{4}\beta \geq \frac{9}{4}.$$

Suppose then that among $\{w_4, x, y\}$, two are at distance 4 from $v$, say $w_4$ and $y$. Then $w_4 = y$ is a 1-vertex, and we denote by $w$ its unique downward neighbor. The local structure is



Two equally worst tables, and the resulting common inequality, are

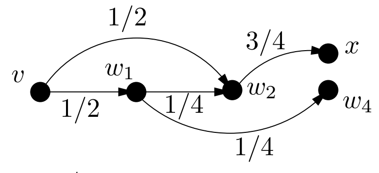
(3.18)

| | $\alpha\Delta_1$ | $\beta\Delta$ |
|---|---|---|
| $1/4x$ | 1 | 5 |
| $1/8z$ | 1 | 6 |
| $1/8t$ | 1 | 7 |
| $1/2w$ | 1 | 5 |

| | $\alpha\Delta_1$ | $\beta\Delta$ |
|---|---|---|
| $1/4x$ | 1 | 6 |
| $1/8z$ | 1 | 6 |
| $1/8t$ | 1 | 5 |
| $1/2w$ | 1 | 5 |

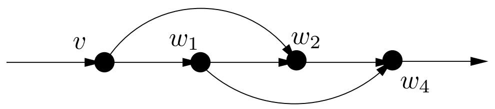
$$\alpha + \frac{43}{8}\beta \geq \frac{11}{4}\ .$$

*Case* 2.c.iii: $w_1$ is a 2-vertex that reaches $w_2$; that is, one of its downward neighbors, say $w_3$, coincides with $w_2$. Then $w_2$ is a 1-vertex, and we denote by $x$ its unique downward neighbor.



$c = 9/4$

A crucial observation is that $x$ cannot be equal to $w_4$. Indeed, if they were equal, then $w_4$ would be a 1-vertex.



In this case, cutting the edge graph $G$ of $P$ at the downward edge emanating from $w_4$ and at the edge entering $v$ would have disconnected $G$, contradicting the fact that $G$ is 3-connected.

We first dispose of the case where $x$ lies lower than $w_4$. The table and inequality are

(3.19)

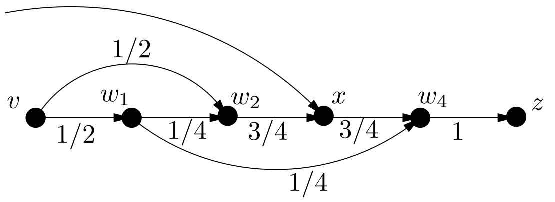
| | $\alpha\Delta_1$ | $\beta\Delta$ |
|---|---|---|
| $3/4x$ | 1 | 4 |
| $1/4w_4$ | 1 | 3 |

$$\alpha + \frac{15}{4}\beta \geq \frac{9}{4}.$$

In what follows we thus assume that $x$ lies above $w_4$.

*Case* 2.c.iii.1: $x$ is a 1-vertex that precedes $w_4$. Suppose first that $w_4$ is the unique downward neighbor of $x$. Then $w_4$ is a 1-vertex, and we denote its unique downward neighbor by $z$. The local structure, table, and inequality are

(3.20)



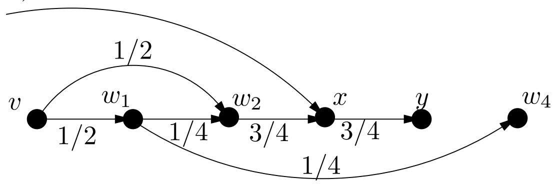
$c = 4$

| | $\alpha\Delta_1$ | $\beta\Delta$ |
|---|---|---|
| $z$ | 3 | 5 |

$$3\alpha + 5\beta \geq 4.$$

Suppose next that the unique downward neighbor $y$ of $x$ is not $w_4$. The local structure, table, and inequality look like this ($y$ is drawn above $w_4$ because this yields a sharper inequality):

(3.21)



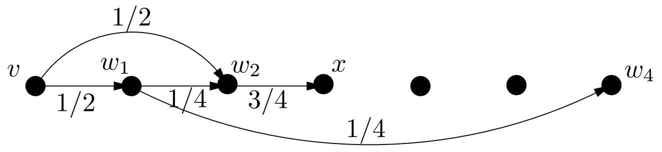
$c = 3$

$$2\alpha + \frac{17}{4}\beta \geq 3.$$

| | $\alpha\Delta_1$ | $\beta\Delta$ |
|---|---|---|
| $3/4y$ | 2 | 4 |
| $1/4w_4$ | 2 | 5 |

*Case* 2.c.iii.2: $x$ is a 2-vertex that precedes $w_4$. This subcase splits into several subcases, where we assume, respectively, that $\Delta(w_4) \geq 6$, $\Delta(w_4) = 4$, and $\Delta(w_4) = 5$.

*Case* 2.c.iii.2(a): Suppose first that $\Delta(w_4) \geq 6$. The configuration looks like this:



$c = 9/4$

The table and inequality are

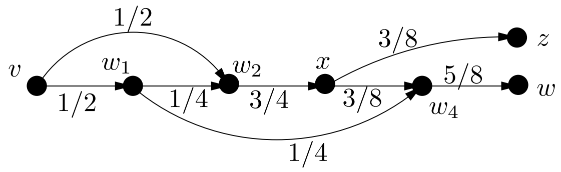
(3.22)

| | $\alpha\Delta_1$ | $\beta\Delta$ |
|---|---|---|
| $3/4x$ | 1 | 3 |
| $1/4w_4$ | 1 | 6 |

$$\alpha + \frac{15}{4}\beta \geq \frac{9}{4}.$$

Note that this is the same inequality as in (3.19).

*Case* 2.c.iii.2(b): Suppose next that $\Delta(w_4) = 4$, and that one of the downward neighbors of $x$ is $w_4$. Let $z$ denote the other downward neighbor. $w_4$ is a 1-vertex, and we denote by $w$ its unique downward neighbor.



$c = 29/8$

The 3-connectivity of the edge graph of $P$ implies, as above, that $w \neq z$. Since we assume that $\Delta(w_4) = 4$, $z$ also lies below $w_4$, and the table and inequality are

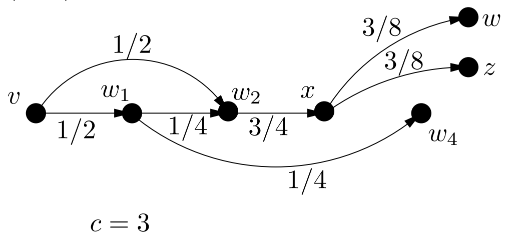
(3.23)

| | $\alpha\Delta_1$ | $\beta\Delta$ |
|---|---|---|
| $5/8w$ | 2 | 5 |
| $3/8z$ | 2 | 6 |

$$2\alpha + \frac{43}{8}\beta \geq \frac{29}{8}.$$

Suppose next that $\Delta(w_4) = 4$ and $w_4$ is not a downward neighbor of $x$. Denote those two neighbors as $w$ and $z$, both of which lie lower than $w_4$, by assumption, and are clearly distinct. The configuration, table, and inequality look like this:

(3.24)

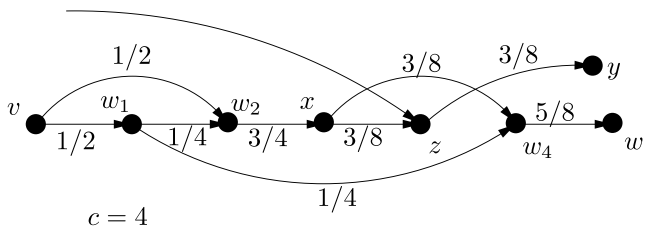


| | $\alpha\Delta_1$ | $\beta\Delta$ |
|---|---|---|
| $1/4w_4$ | 1 | 4 |
| $3/8w$ | 1 | 5 |
| $3/8z$ | 1 | 6 |

$$\alpha + \frac{41}{8}\beta \geq 3.$$

*Case* 2.c.iii.2(c): It remains to consider the case $\Delta(w_4) = 5$. Let $z$ denote the unique vertex lying between $x$ and $w_4$. We may assume that $z$ is connected to $x$, for otherwise $z$ is not reachable from $v$, and we might as well reduce this case to the case $\Delta(w_4) = 4$ just treated.

Consider first the subcase where the other downward neighbor of $x$ is $w_4$ itself. Then $w_4$ is a 1-vertex, and we denote by $w$ its unique downward neighbor. This subcase splits further into two subcases: First, assume that $z$ is a 1-vertex, and let $y$ denote its unique downward neighbor. Clearly, $y$ must lie below $w_4$ (it may coincide with or precede $w$). The configuration looks like this:



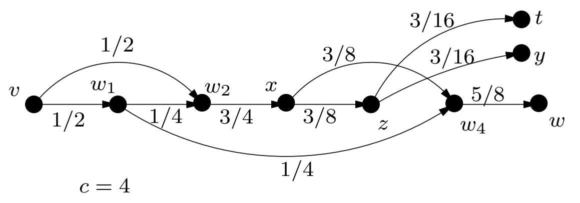
The table and inequality are

(3.25)

| | $\alpha\Delta_1$ | $\beta\Delta$ |
|---|---|---|
| $3/8y$ | 3 | 6 |
| $5/8w$ | 3 | 6 |

$$3\alpha + 6\beta \geq 4.$$

In the other subcase, $z$ is a 2-vertex; we denote its two downward neighbors as $y$ and $t$. The vertices $w, y, t$ all lie below $w_4$ and may appear there in any order (except that $w \neq t$). The configuration looks like this:



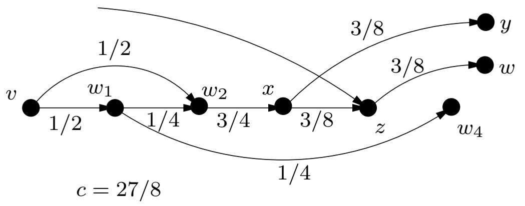
The table and inequality are

(3.26)

| | $\alpha\Delta_1$ | $\beta\Delta$ |
|---|---|---|
| $3/16y$ | 2 | 6 |
| $3/16t$ | 2 | 7 |
| $5/8w$ | 2 | 6 |

$$2\alpha + \frac{99}{16}\beta \geq 4.$$

Consider next the subcase where $w_4$ is not a downward neighbor of $x$. Denote the other downward neighbor of $x$ as $y$, which lies strictly below $w_4$. This subcase splits into three subcases. First, assume that $z$ is a 1-vertex, and denote its unique downward neighbor as $w$. The configuration looks like this:



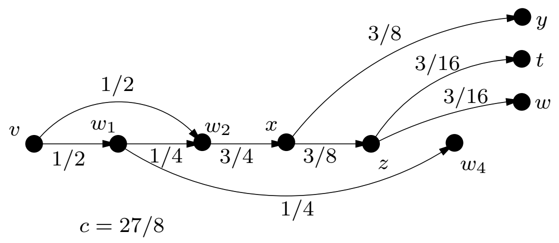
The table and inequality are

(3.27)

| | $\alpha\Delta_1$ | $\beta\Delta$ |
|---|---|---|
| $1/4w_4$ | 2 | 5 |
| $3/8y$ | 2 | 6 |
| $3/8w$ | 2 | 5 |

$$2\alpha + \frac{43}{8}\beta \geq \frac{27}{8}.$$

Second, assume that $z$ is a 2-vertex, so that none of its two downward neighbors is $w_4$. Denote these neighbors as $w$ and $t$. All three vertices $y, t, w$ lie strictly below $w_4$, and $w \neq t$. The configuration looks like this:



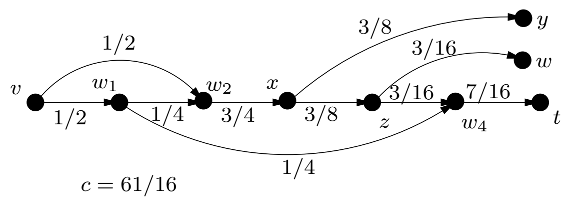
The table and inequality are

(3.28)

| | $\alpha\Delta_1$ | $\beta\Delta$ |
|---|---|---|
| $1/4w_4$ | 1 | 5 |
| $3/8y$ | 1 | 6 |
| $3/16w$ | 1 | 6 |
| $3/16t$ | 1 | 7 |

$$\alpha + \frac{95}{16}\beta \geq \frac{27}{8}.$$

Finally, assume that $z$ is a 2-vertex, so that one of its two downward neighbors is $w_4$. Denote the other neighbor as $w$. In this case $w_4$ is a 1-vertex, and we denote its unique downward neighbor as $t$. All three vertices $y, t, w$ lie strictly below $w_4$. The configuration looks like this:



The table and inequality are

(3.29)

| | $\alpha\Delta_1$ | $\beta\Delta$ |
|---|---|---|
| $3/8y$ | 2 | 6 |
| $3/16w$ | 2 | 7 |
| $7/16t$ | 2 | 6 |

$$2\alpha + \frac{99}{16}\beta \geq \frac{61}{16},$$

which, by the way, is weaker than (3.26).

This completes the case distinction. Thus (3.4) holds for every pair $(\alpha, \beta)$ that satisfies (3.6)–(3.29). In particular, it holds for the pair $(\alpha, \beta) = (\frac{46}{87}, \frac{42}{87})$, which (as discussed at the beginning of the proof) yields the upper bound $\frac{130}{87} < 1.4943$ on the linearity coefficient of random edge. $\square$

**3.3. Discussion.** (1) The analysis has used (twice) the fact that $G$ is a 3-connected graph. Without this assumption, the linearity coefficient becomes 13/8: A lower bound construction can be derived from the figure shown in Case 2.c.iii, and an upper bound can be obtained along the same lines of the preceding proof, using a much shorter case analysis. It is interesting that the proof did not use at all the planarity of the polytope graph $G$.

(2) In an earlier phase of our work, we obtained the upper bound of 3/2 on the linearity coefficient, using a similar but considerably shorter case analysis. Unfortunately, the lengthier case distinction presented in the proof above is not just a refinement of that shorter one (which is the reason for presenting only the lengthier proof). The proof indicates that the problem probably is far from admitting a clean and simple solution—at least using this approach. Of course, it would be interesting to find an alternative simpler way of attacking the problem.

(3) The solution $(\alpha, \beta) = (\frac{46}{87}, \frac{42}{87})$ satisfies (3.9) and (3.20) with equality. If we examine the configuration corresponding to (3.9) and expand it further, we can replace (3.9) by better inequalities, which result in a (slightly) improved bound on the linearity coefficient, at the cost of lengthening further our case analysis. This refinement process can continue for a few more steps, as we have verified. We have no idea whether this iterative refinement process ever converges to some critical configuration, whose further expansion does not improve the bound, and which is then likely to yield a tight bound on the linearity coefficient.

## 4. Other pivot rules.

**4.1. Bland's rule.** For Bland's *least index* pivot rule [2] the facets (inequalities) are numbered. At every nonminimal vertex the rule then dictates choosing the edge that leaves the facet with the smallest number. (A special feature of Bland's rule is that it does not admit cycling even on degenerate programs/nonsimple polytopes, when our geometric description of the rule is, however, not applicable.)

PROPOSITION 4.1. *The linearity coefficient of Bland's rule is* 2.

*Proof.* Figure 2.1 illustrates a family of 3-dimensional LPs on which Bland's rule, started at $v_{\text{start}} = v_{2n-6}$, visits all but one of the vertices. (As we have already

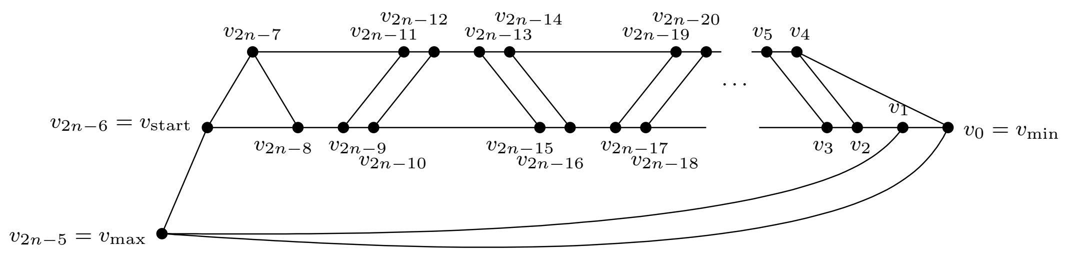


FIG. 4.1. *Lower bound for the greatest decrease rule. All edges are oriented from left to right.*

noted, the directed graph in the figure is readily verified to satisfy the conditions of Theorem 2.2.) Specifically, choose an initial numbering of the facets, where the largest index is assigned to facet $f$. When starting at the vertex $v_{\text{start}} = v_{2n-6}$, the simplex algorithm with Bland's rule visits the $2n-5$ vertices $v_{2n-6}, \dots, v_0$. ☐

**4.2. Dantzig's rule.** *Dantzig's rule* is the original rule proposed by Dantzig when he invented the simplex algorithm. In his setting of a maximization problem formulated in the language of simplex tableaus, the rule requires pivoting into the basis the variable that has the largest reduced cost coefficient (if no variable has positive reduced cost, the current tableau is optimal).

By suitably scaling the inequalities of the LP, Dantzig's rule follows the same path as Bland's rule; see Amenta and Ziegler [1, Observation 2.6]. Thus Dantzig's rule cannot be faster than Bland's rule, and Proposition 4.1 thus implies the following.

PROPOSITION 4.2. *The linearity coefficient of Dantzig's rule is* 2.

**4.3. Greatest decrease rule.** The *greatest decrease* rule moves from any non-optimal vertex to the neighbor with the smallest objective function value. We assume that the objective function is generic, so the vertex is unique. However, the greatest decrease rule may compare nonadjacent neighbors, so the information given by the directed graph is not sufficient to implement it; we rather need explicit objective function values.

PROPOSITION 4.3. *The linearity coefficient of the greatest decrease rule is* $\frac{3}{2}$.

*Proof.* First we show that $\Lambda(\text{GD}) \geq \frac{3}{2}$. Figure 4.1 indicates a family of 3-dimensional LPs. By Theorem 2.2, there is a realization of these LPs with the objective function linear ordering on the vertices given by the left-to-right ordering in our figure. Started at $v_{\text{start}} = v_{2n-6}$, the greatest decrease rule visits all 1-vertices, the global sink, and half of the 2-vertices. Thus it needs $\frac{3}{2}(n-3)$ pivot steps to reach $v_{\min} = v_0$.

For the proof of $\Lambda(\text{GD}) \leq \frac{3}{2}$, we consider an arbitrary instance with $n$, $P$, $\varphi$, and $v_{\text{start}}$ as above. Denote by $n_1$ and $n_2$ the number of visited 1- and 2-vertices, respectively. Thus there are $n-3-n_1$ and $n-3-n_2$ unvisited 1-vertices and 2-vertices, respectively. For every visited 2-vertex $v$ only one of the two direct successors $v'$ and $v''$ is visited. Assuming that $\varphi(v') > \varphi(v'')$, the greatest decrease rule will proceed directly from $v$ to $v''$ and thus skip $v'$, whose objective function value satisfies $\varphi(v) > \varphi(v') > \varphi(v'')$. Thus there is an unvisited vertex uniquely associated with every visited 2-vertex. Thus $n_2 \leq 2n-6-n_1-n_2$, which is equivalent to $n_1 + 2n_2 \leq 2n-6$. We get

$$n_1 + n_2 \;=\; \frac{1}{2}n_1 + \frac{1}{2}(n_1 + 2n_2) \;\leq\; \frac{1}{2}(n-3) + \frac{1}{2}(2n-6) \;\leq\; \frac{3}{2}(n-3).$$

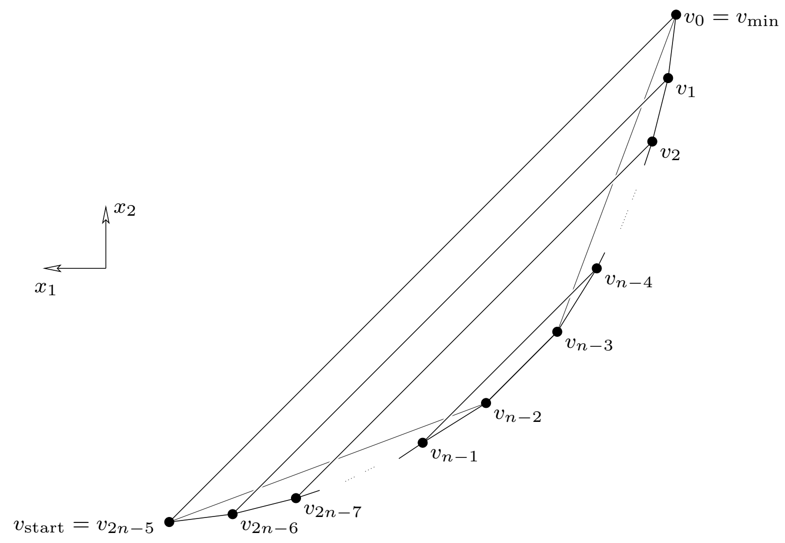


FIG. 4.2. *Lower bounds for the steepest decrease and shadow vertex rules. Planar projection of the polytope: The objective function is $x_1$; it directs all edges from left to right.*

This yields $\Lambda(\text{GD}) \le \frac{3}{2}$ and completes the proof. ∎

**4.4. Steepest decrease rule.** At any nonminimal vertex $v$ the *steepest decrease* pivot rule moves to the neighbor $w$ with $vw$ being the steepest decreasing edge, that is, such that $\frac{\langle c, w-v\rangle}{\|w-v\|\,\|c\|}$ is minimal (where $\langle c, x\rangle$ is the objective function).

PROPOSITION 4.4. *The linearity coefficient of the steepest decrease rule is* 2.

*Proof.* Figure 4.2 depicts a planar projection onto the $(x_1, x_2)$-plane of an LP that is easily constructed either "by hand" or as a deformed product (see Amenta and Ziegler [1]). If the polytope is scaled to be very flat in the $x_3$-direction, then steepest decrease tells the simplex algorithm to use the edge that in the projection has the smallest slope (in absolute value). Thus starting at $v_{\text{start}} = v_{2n-5}$, the steepest decrease rule visits *all* the vertices. ∎

**4.5. Shadow vertex rule.** The *shadow vertex* pivot rule chooses a sequence of edges that lie on the boundary of the 2-dimensional projection of the polytope given by $x \mapsto (\langle c, x\rangle, \langle d, x\rangle)$, where $\langle c, x\rangle$ is the given objective function and $\langle d, x\rangle$ is an objective function that is constructed to be optimal at the starting vertex $v_{\text{start}}$. The vertices that are visited on the path from $v_{\text{start}}$ to $v_{\min}$ are then optimal for objective functions that interpolate between $\langle d, x\rangle$ and $\langle c, x\rangle$. (This pivot rule is known to be polynomial on "random linear programs" in specific models; cf. Borgwardt [3], Ziegler [21], and Spielman and Teng [18].)

PROPOSITION 4.5. *The linearity coefficient of the shadow vertex rule is* 2.

*Proof.* We reuse the LPs of Proposition 4.4/Figure 4.2. Here $v_{2n-5} = v_{\max}$ is optimal for the starting objective function $\langle d, x\rangle = x_2$, while $v_0$ is optimal for $\langle c, x\rangle = x_1$. On the way from $v_{2n-5}$ to $v_{\min} = v_0$ the shadow vertex rule visits *all* the vertices. ∎

**4.6. Random facet.** The *random facet* pivot rule, due to Kalai [10, p. 228], is as follows:

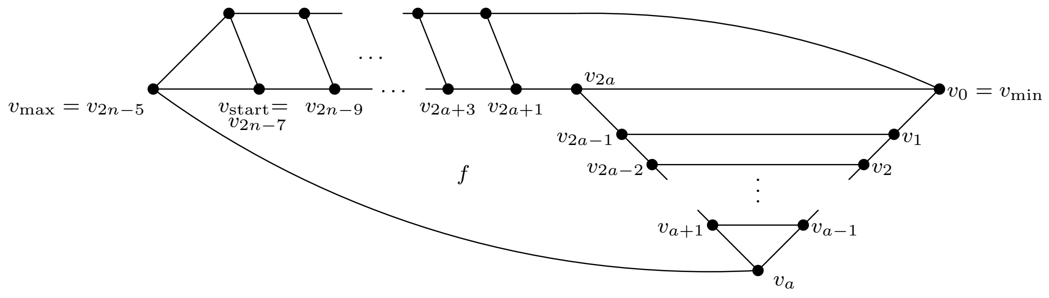


FIG. 4.3. *Lower bound for the random facet rule* (RF). *All edges are oriented from left to right.*

(RF) At any nonoptimal vertex $v$ choose one facet $f$ containing $v$ uniformly at random, and solve the problem restricted to $f$ by applying (RF) recursively. The recursion will eventually restrict to a 1-dimensional subproblem (that is, an edge), which is solved by following the edge.

The 1-dimensional base case singled out here is only implicit in Kalai's work. This is probably the reason why there are different versions of this rule in the literature, which unfortunately were not distinguished. They all differ in the way that 1-vertices are treated. Since the (unique) out-edge of a 1-vertex is always taken with probability one (regardless of which facets we restrict to) we could use the following alternative formulations of the random facet rule:

(RF1) At each nonoptimal vertex $v$ follow the (unique) outgoing edge if $v$ is a 1-vertex. Otherwise choose one facet $f$ uniformly at random containing $v$, and solve the problem restricted to $f$ by applying (RF1) recursively.

(RF2) At any nonoptimal vertex $v$ choose one facet $f$ containing $v$ uniformly at random, and solve the problem restricted to $f$ by applying (RF2) recursively. The minimal vertex opt$(f)$ of $f$ is a 1-vertex, and we follow the (unique) outgoing edge of the vertex opt$(f)$.

The variant (RF1) appears in Gärtner, Henk, and Ziegler [6, p. 350], while the version (RF2) is from Gärtner [5], who, however, formulated this variant of the random facet rule for combinatorial cubes, where the formulations above are equivalent.

Note that (RF) uses randomness at every vertex, and (RF1) would follow a path of 1-vertices deterministically, while (RF2) takes at most one deterministic step in a row. This results in distinct pivot rules, with different worst case examples.

PROPOSITION 4.6. *For each version* (RF), (RF1), *and* (RF2) *of the random facet rule, the linearity coefficient is* 2.

*Proof.* Figure 4.3 depicts a family of LPs with $2n - 4 = 2a + 2b + 2$ vertices and $n = a + b + 3$ facets. For each of the $b$ 1-vertices $v_{\text{start}} = v_{2n-7}, v_{2n-9}, \ldots, v_{2a+1}$, the probability of leaving it via choosing facet $f$ is $\frac{1}{2}$. After choosing facet $f$, (RF) "sticks" to facet $f$ until $v_a$ is reached.

Choosing $a = k^2$ and $b = k$, we obtain a family of LPs with $n = k^2 + k + 3$ facets. Then (RF) sticks to facet $f$ with probability $p \geq 1 - (\frac{1}{2})^k$. Thus the expected number of visited vertices is at least

$$\left(1 - \left(\frac{1}{2}\right)^k\right) (2a + b) \;\geq\; 2k^2 - \frac{2k^2}{2^k}.$$

Since there are $n = k^2 + k + 3$ facets, the linearity coefficient is 2.

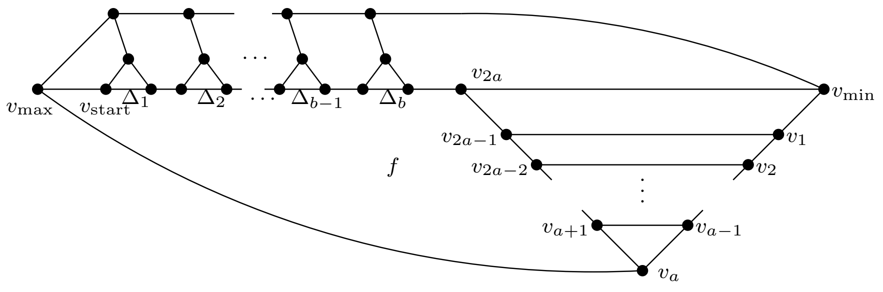


FIG. 4.4. *Lower bounds for the* (RF1) *and* (RF2) *variants of the random facet rule, and for the least entered rule with random edge as the tie-breaking rule. All edges are oriented from left to right.*

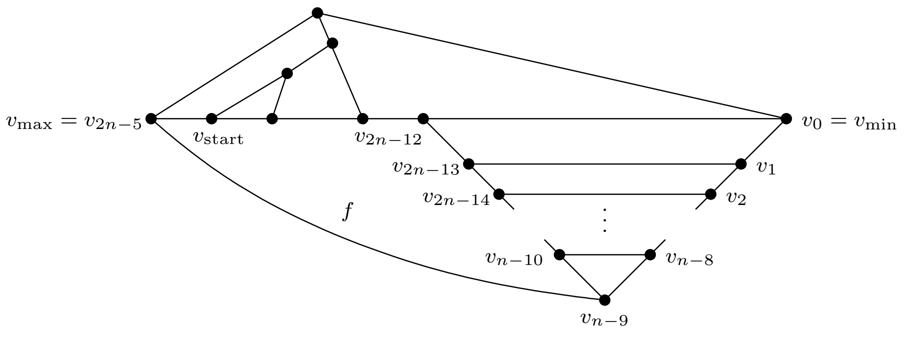


FIG. 4.5. *Lower bound for the least entered rule with greatest decrease as the tie-breaking rule. All edges are oriented from left to right.*

The version (RF1) of the random facet rule follows the path of 1-vertices $v_{\text{start}} = v_{2n-7}, v_{2n-9}, \dots, v_{2a+1}$ deterministically. We can cut off each of these vertices. This yields the graphs depicted in Figure 4.4. At each source of the new facets $\Delta_1, \dots, \Delta_b$, the facet $f$ is chosen with probability $\frac{1}{3}$. If any of the other two facets is chosen, we end up at the sink vertex of the respective facet $\Delta_i$. Thus the linearity coefficient remains 2, and only the rate of convergence decreases. The same works for (RF2) as well. ▯

**4.7. Least entered rule.** At any nonoptimal vertex, the *least entered* pivot rule chooses the decreasing edge that leaves the facet that has been left least often in the previous moves. In case of ties, a tie-breaking rule is used to determine the decreasing edge to be taken. Any other pivot rule can be used as a tie-breaking rule.

The least entered rule was first formulated by Norman Zadeh around 1980 (see [13] and [21]). It has still not been determined whether Zadeh's rule is polynomial if the dimension is part of the input. Zadeh has offered $1000 for solving this problem.

PROPOSITION 4.7. *The linearity coefficient of the least entered rule with greatest decrease as tie-breaking rule is* 2.

*Proof.* Figure 4.5 describes a family of 3-dimensional LPs, where the left-to-right ordering of the vertices suggested by the figure can be realized, according to the Mihalisin–Klee theorem (Theorem 2.2). Starting at $v_{\text{start}} = v_{2n-6}$, the greatest decrease rule decides to leave the facet $f$. Following two 1-vertices, the facet $f$ is entered again. All upcoming facets have not been visited before. Thus the least entered rule "sticks" to the facet $f$, and $2n - 7$ vertices (that is, all but 3 vertices) are visited. ∎

PROPOSITION 4.8. *The linearity coefficient of the least entered rule with random edge as the tie-breaking rule is* 2.

*Proof.* Figure 4.4 describes LPs with $2n-4 = 2a+4b+2$ vertices and $n = a+2b+3$ facets. At the sources of the facets, $\Delta_i$, the random edge rule leaves the facet $f$ with probability $\frac{1}{2}$. As soon as $f$ is left once, it will be revisited, and the least entered rule will "stick" to the facet $f$. (Thus the only way not to "stick" to $f$ is that the random edge rule chooses to continue along $f$ until it reaches the vertex $v_{2a}$.) When the least entered rule "sticks" to the facet $f$, all of the $2a$ vertices $v_{2a-1}, v_{2a-2}, \ldots, v_1, v_0$ are visited.

Now the analysis is exactly the same as in the proof of Proposition 4.6. Thus choosing $a = k^2$ and $b = k$ yields that the linearity coefficient is 2. ∎

**Acknowledgments.** We are grateful to Emo Welzl and Günter Rote for inspiring discussions and helpful comments, and to Gabriel Nivasch for pointing out an error in an earlier version of the proof of Theorem 3.2.

## REFERENCES

[1] N. AMENTA AND G. M. ZIEGLER, *Deformed products and maximal shadows*, in Advances in Discrete and Computational Geometry (South Hadley, MA, 1996), B. Chazelle, J. E. Goodman, and R. Pollack, eds., Contemp. Math. 223, AMS, Providence, RI, 1998, pp. 57–90.

[2] R. G. BLAND, *New finite pivoting rules for the simplex method*, Math. Oper. Res., 2 (1977), pp. 103–107.

[3] K. H. BORGWARDT, *The Simplex Method. A Probabilistic Analysis*, Algorithms and Combinatorics 1, Springer-Verlag, Berlin, Heidelberg, 1987.

[4] G. B. DANTZIG, *Linear Programming and Extensions*, Princeton University Press, Princeton, NJ, 1963.

[5] B. GÄRTNER, *Combinatorial linear programming: Geometry can help*, in Randomization and Approximation Techniques in Computer Science (Barcelona, 1998), Lecture Notes in Comput. Sci. 1518, Springer, Berlin, 1998, pp. 82–96.

[6] B. GÄRTNER, M. HENK, AND G. M. ZIEGLER, *Randomized simplex algorithms on Klee-Minty cubes*, Combinatorica, 18 (1998), pp. 349–372.

[7] B. GÄRTNER, J. SOLYMOSI, F. TSCHIRSCHNITZ, P. VALTR, AND E. WELZL, *One line and n points*, in Proceedings of the 33rd ACM Symposium on the Theory of Computing, Crete, Greece, 2C, ACM Press, New York, 2001, pp. 306–315.

[8] M. JOSWIG, V. KAIBEL, AND F. KÖRNER, *On the k-systems of a simple polytope*, Israel J. Math., 129 (2002), pp. 109–117.

[9] G. KALAI, *A subexponential randomized simplex algorithm*, in Proceedings of the 24th ACM Symposium on the Theory of Computing, Victoria, BC, 1992, ACM Press, New York, 1992, pp. 475–482.

[10] G. KALAI, *Linear programming, the simplex algorithm and simple polytopes*, Math. Programming, 79 (1997), pp. 217–233.

[11] D. G. KELLY, *Some results on random linear programs*, Methods Oper. Res., 40 (1981), pp. 351–355.

[12] V. KLEE, *Paths on polyhedra.* I, J. SIAM, 13 (1965), pp. 946–956.

[13] V. KLEE AND P. KLEINSCHMIDT, *The d-step conjecture and its relatives*, Math. Oper. Res., 12 (1987), pp. 718–755.

[14] J. MATOUŠEK, M. SHARIR, AND E. WELZL, *A subexponential bound for linear programming*, Algorithmica, 16 (1996), pp. 498–516.

[15] J. MATOUŠEK AND T. SZABÓ, *Random edge can be exponential on abstract cubes*, in Proceedings of the 45th Annual IEEE Symposium on Foundations of Computer Science, 2004, pp. 92–1000.

[16] R. MECHTEL, *Randomized Pivot Rules for the Simplex Algorithm on Three-Dimensional Problems*, Diplomarbeit, TU Berlin, 2003.

[17] J. MIHALISIN AND V. KLEE, *Convex and linear orientations of polytopal graphs*, Discrete Comput. Geom., 24 (2000), pp. 421–435.

[18] D. Spielman and S.-H. Teng, *Smoothed analysis of algorithms: Why the simplex algorithm usually takes polynomial time*, J. ACM, 51 (2004), pp. 385–463.

[19] C. A. Tovey, *Low order polynomial bounds on the expected performance of local improvement algorithms*, Math. Programming, 35 (1986), pp. 193–224.

[20] G. M. Ziegler, *Lectures on Polytopes*, Graduate Texts in Math. 152, Springer-Verlag, New York, 1995; “Updates, Corrections, and More” at www.math.tu-berlin.de/~ziegler.

[21] G. M. Ziegler, *Typical and extremal linear programs*, in The Sharpest Cut: The Impact of Manfred Padberg and His Work, M. Grötschel, ed., MPS-SIAM Ser. Optim. 4, SIAM, Philadelphia, PA, 2004, pp. 217–230.

SIAM J. COMPUT. © 2005 Society for Industrial and Applied Mathematics
Vol. 34, No. 2, pp. 498–513

# TEMPORAL REASONING ABOUT TWO CONCURRENT SEQUENCES OF EVENTS[*]

YASUNORI ISHIHARA[†], SHIN ISHII[‡], HIROYUKI SEKI[‡], AND MINORU ITO[‡]

**Abstract.** This paper discusses temporal reasoning with respect to constraints on two concurrent sequences of events. If two given sequences of events can be mapped into one sequence that satisfies a given constraint, then the constraint is said to be consistent. First, we mention that the consistency of such constraints is NP-complete. Then we introduce the notion of graph representations of constraints. If a graph representation of a given constraint $c$ can be constructed in polynomial time, then the consistency of $c$ is decidable in polynomial time. However, it is shown that the graph representability of a given $c$ is coNP-complete. Next, we propose a subclass CDC$^{\neq}$ of constraints such that for each constraint $c$ in CDC$^{\neq}$, a graph representation of $c$ can be constructed in polynomial time. The expressive power of CDC$^{\neq}$ is incomparable to any other subclasses of constraints for which the consistency problem is known to be tractable.

**Key words.** temporal reasoning, temporal constraint, consistency, complexity

**AMS subject classifications.** 68Q25, 68Q20

**DOI.** 10.1137/S0097539799362172

## 1. Introduction.

In many practical information systems, each fragment of data has temporal information. For example, in relational databases, relations are often augmented by temporal attributes (such databases are called temporal databases [1]). It is often desirable for information systems to manage such temporal information in an intelligent way. For example, suppose that a knowledge base system has the following information:

- Last night, the people in the restaurant heard two shots.
- The electricity was off at least between the first and second shots.
- After the electric power resumed, the people found all the money in the restaurant had been stolen.

We want the knowledge base system to infer that the people found all the money stolen *after* the second shot. This is a trivial but typical example of *temporal reasoning*.

In this paper, temporal reasoning about two concurrent sequences of events is considered. Two sets of time variables $S = \{s_0, s_1, \dots, s_m\}$ and $T = \{t_0, t_1, \dots, t_n\}$ are used for describing temporal constraints, where $s_0, s_1, \dots, s_m$ represent time points of one local clock and $t_0, t_1, \dots, t_n$ represent time points of the other clock. A temporal constraint consists of expressions of the forms $s_i < t_j$, $s_i > t_j$, $s_i \leq t_j$, $s_i \geq t_j$, $s_i = t_j$, and $s_i \neq t_j$, and Boolean operators $\neg$, $\vee$, and $\wedge$. One of the applications of temporal reasoning about such constraints is belief revision [4] in a multiagent environment, as shown in the next example. As far as we know, no paper has focused on the class of constraints such that the number of local clocks is fixed.

*Example* 1.1. Consider the following multiagent environment: Each agent has its own local clock and records its observations, each of which is a pair consisting

[*]Received by the editors September 27, 1999; accepted for publication (in revised form) August 18, 2004, published electronically February 3, 2005.
http://www.siam.org/journals/sicomp/34-2/36217.html

[†]Graduate School of Information Science and Technology, Osaka University, Suita, Osaka, 565-0871 Japan (ishihara@ist.osaka-u.ac.jp).

[‡]Graduate School of Information Science, Nara Institute of Science and Technology, Ikoma, Nara, 630-0192 Japan (ishii@is.naist.jp, seki@is.naist.jp, ito@is.naist.jp).

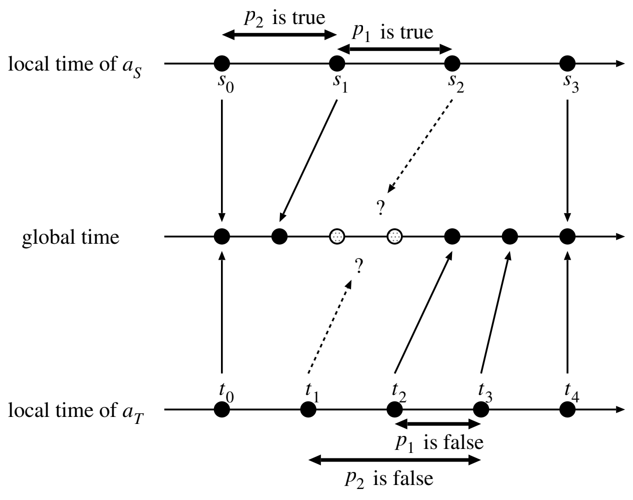


FIG. 1. *Example of temporal reasoning about two concurrent sequences of events.*

of a proposition and an observation time. The agents sometimes meet together and exchange their observations in order to revise and refine their information on the observation times.

See Figure 1. $s_0, \ldots, s_3$ represent local time points of agent $a_S$ with $s_0 < \cdots < s_3$, and $t_0, \ldots, t_4$ represent local time points of agent $a_T$ with $t_0 < \cdots < t_4$. The agents met together at $s_0 = t_0$ and $s_3 = t_4$. Suppose that $a_S$ observed that a proposition $p_1$ was true during its local time interval $[s_1, s_2]$. Also suppose that $a_T$ observed that $p_1$ was false during $[t_2, t_3]$. Then the agents can conclude that intervals $[s_1, s_2]$ and $[t_2, t_3]$ are disjoint, i.e., $(s_1 \geq t_3) \vee (s_2 \leq t_2)$. In this paper, a constraint of this form is called a *disjointness constraint* and is denoted by $[s_1, s_2] \not\cap [t_2, t_3]$. The agents also obtain $[s_0, s_1] \not\cap [t_1, t_3]$ from the observations of $p_2$.

Now, several facts can be inferred from the obtained constraint $c = ([s_1, s_2] \not\cap [t_2, t_3]) \wedge ([s_0, s_1] \not\cap [t_1, t_3])$. For example, $s_1 \leq t_1$ can be concluded since $c \wedge (s_1 > t_1)$ is inconsistent (i.e., unsatisfiable). On the other hand, the order between $s_2$ and $t_1$ cannot be determined since all of $c \wedge (s_2 < t_1)$, $c \wedge (s_2 = t_1)$, and $c \wedge (s_2 > t_1)$ are consistent.

Another possible application is job scheduling for two processors, where each job is denoted by a time interval and each pair of mutually exclusive jobs is specified by a disjointness constraint.

The contribution of this paper is as follows. We first mention that the consistency of constraints on two local clocks is NP-complete when both conjunction and disjunction are freely used. Next, we introduce the notion of *graph representations* of constraints. A graph representation of a constraint $c$ is a directed graph representing all the valuations that satisfy $c$. If $c$ has a graph representation $G_c$, and $G_c$ can be constructed in polynomial time, then the consistency of $c$ is also decidable in polynomial time. However, it is shown that the graph representability of a given $c$ is coNP-complete (and therefore constructing $G_c$ is coNP-hard). Next, we propose a new tractable subclass CDC$^{\neq}$ of constraints. CDC$^{\neq}$ stands for conjunctive disjointness constraints with inequalities, and it can express conjunctions of constraints in the form of $[s_i, s_{i'}] \not\cap [t_j, t_{j'}]$ or $s_i \,\Theta\, t_j$, where $\Theta \in \{<, >, \leq, \geq, =, \neq\}$. All the constraints appearing in Example 1.1 are expressible in CDC$^{\neq}$. We show that for each constraint $c$ in CDC$^{\neq}$, a graph representation of $c$ can be constructed in polynomial time. Let $m$ be the number of time variables of one of the two local clocks, and $n$ the other clock. The consistency of $c \in$ CDC$^{\neq}$ is decidable in $O(|c|mn)$ time, where $|c|$ is the size of $c$. Lastly, we show the intractability of constraints generated by disjointness constraints and conjunction and disjunction operators. For such general disjointness constraints, the consistency is NP-complete and the graph representability is coNP-complete.

The rest of the paper is organized as follows. Constraints on two clocks are formulated in section 2. In section 3, graph representations are introduced. It is also shown that the graph representability of a given constraint is coNP-complete. In section 4, a tractable subclass CDC$^{\neq}$ of constraints is proposed. In section 5, the intractability of general disjointness constraints is shown. Section 6 compares the expressive powers of the classes of constraints. Section 7 summarizes the paper.

## 2. Constraints on two clocks.

Let $R$ be an infinite set of *global time points.* Suppose that a total order $\leq$ is defined on $R$. $r \leq r'$ means that point $r$ precedes or is equal to $r'$. When $r \leq r'$ and $r \neq r'$, we write $r < r'$.

Let $S = \{s_0, s_1, \ldots, s_m\}$ and $T = \{t_0, t_1, \ldots, t_n\}$ $(m,\ n \geq 1)$ be sets of *variables.* We write $s_i \leq_S s_j$ and $t_i \leq_T t_j$ if $i \leq j$, and we write $s_i <_S s_j$ and $t_i <_T t_j$ if $i < j$. Intuitively, $S$ and $T$ are sets of *local time points.*

Let $\Sigma_{ST}$ be the family of all the *valuations* $\sigma : S \cup T \to R$ satisfying the following conditions:

- $\sigma(s_0) = \sigma(t_0)$;
- $\sigma(s_m) = \sigma(t_n)$;
- if $s <_S s'$, then $\sigma(s) < \sigma(s')$;
- if $t <_T t'$, then $\sigma(t) < \sigma(t')$.

The first two conditions are introduced merely for theoretical simplicity. Namely, instead of the first condition, we can put dummy variables $s_{-\infty}$ and $t_{-\infty}$ such that $s_{-\infty} <_S s_0$, $t_{-\infty} <_T t_0$, and $\sigma(s_{-\infty}) = \sigma(t_{-\infty})$. On the other hand, the last two conditions are essential. $\sigma$ must preserve the temporal orders of local time points.

Hereafter, we do not distinguish isomorphic valuations with respect to $<$ and $=$. In other words, we are interested in only the quotient sets of $\Sigma_{ST}$ under $<$ and $=$. Therefore, $\sigma$ will be regarded as a permutation of $S \cup T$ which is consistent with both $<_S$ and $<_T$ (although it may hold that $\sigma(s) = \sigma(t)$ for some $s \in S$ and $t \in T$), and $\Sigma_{ST}$ will be regarded as the family of such permutations.

An *atomic constraint* is an expression with one of the following forms: $s < t$, $s > t$, $s \leq t$, $s \geq t$, $s = t$, and $s \neq t$. A *constraint* is generated from atomic constraints and Boolean operators $\neg$, $\vee$, and $\wedge$. For readability, we may use notation such as $t < s < t'$ to mean $(s > t) \wedge (s < t')$. The *satisfaction relation* is defined in an ordinary way, and we write $\sigma \models c$ (read as $\sigma$ *satisfies* $c$) if $c$ is true under valuation $\sigma$. If $\sigma \models c$ for some $\sigma$, then $c$ is *consistent* (or *satisfiable*). By $c \models c'$, we mean that every valuation $\sigma \in \Sigma_{ST}$ satisfying $c$ also satisfies $c'$. We say that $c$ is *equivalent* to $c'$, denoted $c \equiv c'$, if both $c \models c'$ and $c' \models c$ hold.

The *consistency problem* is to determine whether, given sets $S$ and $T$ of variables and constraint $c$, $c$ is consistent or not. The *implication problem* is to determine whether $c \models c'$ holds or not for given sets $S$ and $T$ of variables and constraints $c$ and $c'$. Since $c \models c'$ if and only if $\neg c \wedge c'$ is inconsistent, we mainly focus on the consistency problem in this paper.

Define the size of $S$ and $T$ as $m$ and $n$, respectively. Also define the size of $c$ as the number of atomic constraints in $c$.

THEOREM 2.1. *The consistency of an arbitrary constraint is in NP. The consistency of a constraint in conjunctive normal form (CNF) is NP-hard.*

*Proof.* The consistency problem is obviously in NP. The NP-hardness is shown by reducing the satisfiability problem of CNF logical formulas to this problem. In the reduction, each logical variable $x_i$ in a given logical formula is replaced with an atomic constraint $s_i < t_i$. Whether $s_i < t_i$ holds or not can be determined independently of other atomic constraints $s_j < t_j$. Thus, the obtained constraint is consistent if and only if the original logical formula is satisfiable. □

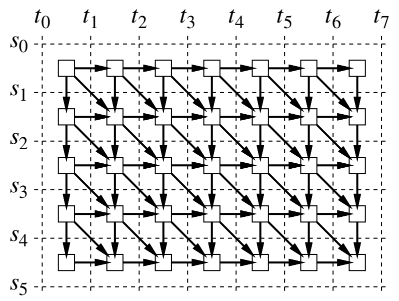


FIG. 2. $G_{ST}$.

## 3. Consistency and graph representability.

**3.1. Graph representations of constraints.** First, we define the graph $G_{ST}$.

DEFINITION 3.1. *Let $S = \{s_0, \ldots, s_m\}$ and $T = \{t_0, \ldots, t_n\}$. Define $G_{ST}$ as a directed acyclic graph consisting of $mn$ nodes arranged in $m$ rows and $n$ columns, with each node having outgoing arcs to the right, lower, and lower-right nodes (if existing). The node at the $i$th row of the $j$th column is denoted by $(i, j)$.*

$G_{ST}$ for $S = \{s_0, \ldots, s_5\}$ and $T = \{t_0, \ldots, t_7\}$ is shown in Figure 2. The $m + 1$ dotted horizontal lines (labeled $s_0, \ldots, s_m$) and $n + 1$ dotted vertical lines (labeled $t_0, \ldots, t_n$) are auxiliary lines explained below.

A *complete path* on $G_{ST}$ is a path from $(1, 1)$ to $(m, n)$. Let $W_{ST}$ denote the set of all the complete paths on $G_{ST}$. There is a one-to-one correspondence between $W_{ST}$ and $\Sigma_{ST}$. To see this, define a mapping $\rho : W_{ST} \to \Sigma_{ST}$ as follows. Let $w \in W_{ST}$.

- $\rho(w)(s_0) = \rho(w)(t_0)$ and $\rho(w)(s_m) = \rho(w)(t_n)$.
- If $w$ crosses line $s_i$ before line $t_j$, then $\rho(w)(s_i) < \rho(w)(t_j)$.
- If $w$ crosses line $t_j$ before line $s_i$, then $\rho(w)(s_i) > \rho(w)(t_j)$.
- If $w$ crosses lines $s_i$ and $t_j$ at the same time, then $\rho(w)(s_i) = \rho(w)(t_j)$.

It can be shown that $\rho$ is a bijection. Let $\rho^{-1}$ denote the inverse of $\rho$.

If $G$ is a subgraph of $G'$, we write $G \subseteq G'$. The union $\cup$ (resp., intersection $\cap$) of subgraphs of $G_{ST}$ is defined as the least upper bound (resp., greatest lower bound) of the subgraphs with respect to $\subseteq$.

The notion of complete paths is extended to subgraphs of $G_{ST}$. That is, for a subgraph $G$ of $G_{ST}$, if there is a path from $(1, 1)$ to $(m, n)$ on $G$, then the path is called a complete path on $G$.

DEFINITION 3.2. *Let $c$ be a constraint and let $G$ be a subgraph of $G_{ST}$. If the following two conditions hold, then $G$ is a* graph representation *of $c$:*

- *$\rho(w) \models c$ for every complete path $w$ on $G$; and*
- *$\rho^{-1}(\sigma)$ is a complete path on $G$ for every valuation $\sigma$ such that $\sigma \models c$.*

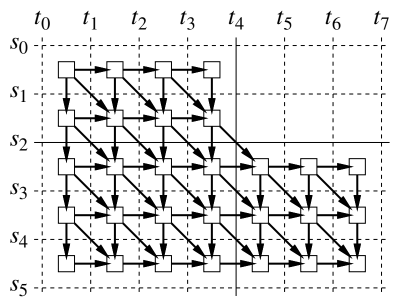

$t_0$ $t_1$ $t_2$ $t_3$ $t_4$ $t_5$ $t_6$ $t_7$
$s_0$ $s_1$ $s_2$ $s_3$ $s_4$ $s_5$

FIG. 3. *Minimum graph representation of $s_2 \leq t_4$.*

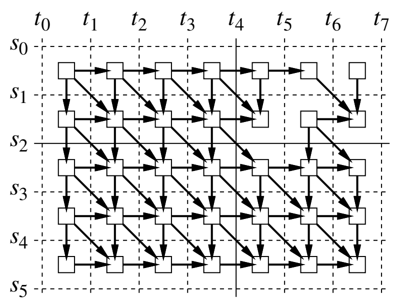

$t_0$ $t_1$ $t_2$ $t_3$ $t_4$ $t_5$ $t_6$ $t_7$
$s_0$ $s_1$ $s_2$ $s_3$ $s_4$ $s_5$

FIG. 4. *Another graph representation of $s_2 \leq t_4$.*

*If $c$ has a graph representation, then $c$ is* graph representable.

Some constraints have more than one graph representation, while some constraints have no graph representation. For example, both Figures 3 and 4 are graph representations of $s_2 \leq t_4$. On the other hand, as will be shown later, $(s_2 \leq t_4) \vee (s_4 \leq t_6)$ has no graph representation. Note that false has a graph representation (e.g., the empty graph).

For a graph-representable constraint $c$, let $G_c$ denote an arbitrary graph representation of $c$.

LEMMA 3.3. *Let $c$ and $c'$ be graph-representable constraints. Then $G_c \cap G_{c'}$ is a graph representation of $c \wedge c'$.*

*Proof.* Any complete path $w$ on $G_c \cap G_{c'}$ is contained in both $G_c$ and $G_{c'}$. Therefore $\rho(w) \models c \wedge c'$. Conversely, consider any valuation $\sigma$ such that $\sigma \models c \wedge c'$. Then $\rho^{-1}(\sigma)$ must be contained in both $G_c$ and $G_{c'}$. Therefore, $\rho^{-1}(\sigma)$ is contained in $G_c \cap G_{c'}$. ☐

Suppose that $c$ has a graph representation $G_c$. By Definition 3.2, $c$ is consistent if and only if $(m, n)$ is reachable from $(1, 1)$ on $G_c$. Since the reachability is decidable in $O(mn)$ time, we obtain the following lemma.

LEMMA 3.4. *Let $c = c_1 \vee \cdots \vee c_l$ be a disjunction of graph-representable constraints $c_1, \ldots, c_l$. Suppose that each $G_{c_i}$ can be constructed in $O(f(c_i, m, n))$ time. Then the consistency of $c$ is decidable in $O(f(c_1, m, n) + \cdots + f(c_l, m, n) + lmn)$ time.*

*Proof.* $c$ is consistent if and only if some $c_i$ is consistent. For each $i$, the consistency of $c_i$ is decidable in $O(f(c_i, m, n) + mn)$ time. Therefore, the consistency of $c$ is decidable in $O(f(c_1, m, n) + \cdots + f(c_l, m, n) + lmn)$ time. ☐

**3.2. Minimum graph representations.** A graph representation may contain redundant nodes and arcs. The following lemma states the existence of the simplest graph representation.

LEMMA 3.5. *Among all the graph representations of $c$, there is a unique minimum graph representation $G^*_c$ with respect to $\subseteq$.*

*Proof.* The number of all the graph representations of $c$ is finite since $G_{ST}$ is a finite graph. Therefore, the intersection of all the graph representations of $c$ can be defined. We show that the intersection $G^*_c$ is the unique minimum graph representation of $c$. By Lemma 3.3, $G^*_c$ is a graph representation of $c$ $(= c \wedge c \wedge \cdots \wedge c)$. Clearly, $G^*_c \subseteq G_c$ for any graph representation $G_c$ of $c$. Hence, $G^*_c$ satisfies the lemma. $\square$

LEMMA 3.6. *Let $c$ be a graph-representable constraint. The minimum graph representation $G^*_c$ of $c$ can be constructed in $O(f(c,m,n)+mn)$ time, where $f(c,m,n)$ is the time complexity of constructing some graph representation of $c$.*

*Proof.* Suppose that a graph representation $G_c$ of $c$ is obtained. Then $G^*_c$ can be constructed by the following algorithm:

1. Mark all the nodes and arcs that are reachable from $(1,1)$ in $G_c$. This can be done in $O(mn)$ time by performing depth first search from $(1,1)$.
2. Mark all the nodes and arcs from which $(m,n)$ is reachable in $G_c$. This can be done in $O(mn)$ time by performing depth first search from $(m,n)$, exploring the arcs in the opposite direction.
3. Excepting the nodes and arcs which are marked both in steps 1 and 2 above, remove the other nodes and arcs.

This algorithm is correct since the resultant graph contains only the nodes and arcs that are contained by a complete path on $G_c$. $\square$

By Lemma 3.3, $G_c \cap G_{c'}$ is a graph representation of $c \wedge c'$. However, $G^*_c \cap G^*_{c'}$ is not necessarily the minimum graph representation of $c \wedge c'$. For example, consider $G^*_c$ and $G^*_{c'}$ which have no common complete paths. Then $G^*_{c \wedge c'}$ is the empty graph, but $G^*_c \cap G^*_{c'}$ is not necessarily empty.

**3.3. Complexity of deciding graph representability.** We are interested in the complexity $f(c,m,n)$ of constructing $G_c$. Unfortunately, the graph representability of a given $c$ with disjunction is coNP-complete (and therefore constructing $G_c$ is coNP-hard), even if $c$ is in disjunctive normal form (DNF). First, we characterize the graph representability.

LEMMA 3.7. *The following properties are equivalent:*

1. *$c$ has no graph representation.*
2. *There is a complete path $w$ on $G_{ST}$ such that*
   - *$\rho(w) \not\models c$, and*
   - *for every arc $a$ contained in $w$, there is another complete path $w_a$ on $G_{ST}$ containing the arc $a$ such that $\rho(w_a) \models c$.*

*Proof.* Suppose that $c$ has no graph representation. Let $G$ be the minimum graph that contains all the complete paths satisfying $c$. Since $c$ has no graph representation, there is a complete path $w$ on $G$ such that $\rho(w) \not\models c$. By the definition of $G$, for every arc $a$ contained in $w$, there is another complete path $w_a$ containing the arc $a$ such that $\rho(w_a) \models c$. Thus the second property holds.

Conversely, suppose that the second property holds. Also assume that $c$ has a graph representation $G_c$. Then $G_c$ does not contain $w$ of the second property, since $G_c$ contains only the complete paths $w'$ such that $\rho(w') \models c$. However, by the second property, for every arc $a$ in $w$, there is $w_a$ containing the arc $a$ such that $\rho(w_a) \models c$. By definition, $G_c$ contains $w_a$. This implies that all the arcs in $w$ must be contained

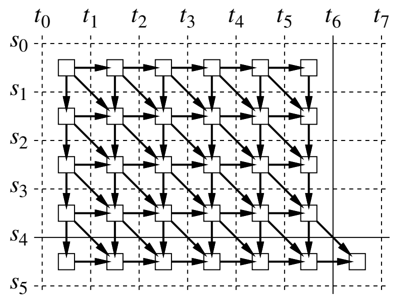

$t_0$ $t_1$ $t_2$ $t_3$ $t_4$ $t_5$ $t_6$ $t_7$
$s_0$ $s_1$ $s_2$ $s_3$ $s_4$ $s_5$

FIG. 5. *Minimum graph representation of* $s_4 \leq t_6$.

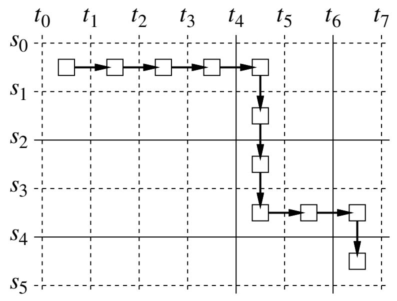

$t_0$ $t_1$ $t_2$ $t_3$ $t_4$ $t_5$ $t_6$ $t_7$
$s_0$ $s_1$ $s_2$ $s_3$ $s_4$ $s_5$

FIG. 6. *A complete path not satisfying* $(s_2 \leq t_4) \vee (s_4 \leq t_6)$.

in $G_c$, and contradicts the assumption that $w$ is not a complete path on $G_c$. Thus, the second property implies the first one. □

We show that $c = (s_2 \leq t_4) \vee (s_4 \leq t_6)$ has no graph representation. The minimum graph representation of $s_4 \leq t_6$ is shown in Figure 5. Consider the complete path $w$ shown in Figure 6. $w$ does not satisfy $c$ since $w$ crosses line $t_4$ before line $s_2$, and crosses line $t_6$ before line $s_4$. However, for each arc $a$ in $w$, there is another complete path $w_a$ containing $a$ such that $\rho(w_a) \models c$ (see Figures 3 and 5). By Lemma 3.7, $c$ has no graph representation.

Before showing the coNP-completeness of the graph representability, we introduce a coNP-complete problem. Let $F$ be a logical formula in DNF such that both $\nu_{\text{true}}$ and $\nu_{\text{false}}$ satisfy $F$, where $\nu_{\text{true}}$ (resp., $\nu_{\text{false}}$) is the interpretation that maps every logical variable to true (resp., false). The *modified tautology problem* is to decide whether such a given logical formula $F$ is a tautology (i.e., $F$ is satisfied by all the interpretations). It is easily shown that the modified tautology problem is coNP-complete.

THEOREM 3.8. *The graph representability of an arbitrary constraint is in coNP. The graph representability of a constraint $c$ with disjunction is coNP-hard, even if $c$ is in DNF.*

*Proof.* The second property of Lemma 3.7 is decidable by an NP algorithm as follows. First, guess a complete path $w$ and verify that $\rho(w) \not\models c$. Then, for every arc $a$ contained in $w$, guess a complete path $w_a$ containing $a$ and verify that $\rho(w_a) \models c$.

To see the coNP-hardness, we reduce the modified tautology problem to the graph representability problem. Let $F = F_1 \vee \cdots \vee F_k$ be a DNF formula with $n$ variables $x_1, \ldots, x_n$ such that both $\nu_{\text{true}}$ and $\nu_{\text{false}}$ satisfy $F$. Let

$$S = \{s_0, \ldots, s_{n+1}\}, \quad T = \{t_0, \ldots, t_{n+1}\}.$$

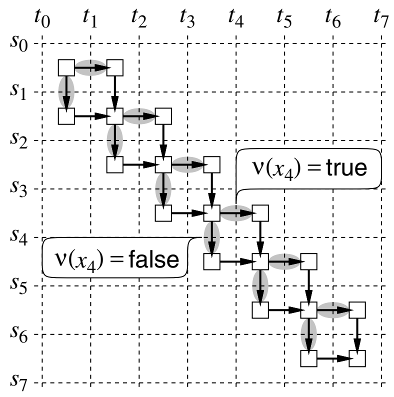

$t_0$ $t_1$ $t_2$ $t_3$ $t_4$ $t_5$ $t_6$ $t_7$
$s_0$ $s_1$ $s_2$ $s_3$ $s_4$ $s_5$ $s_6$ $s_7$
$\nu(x_4) = \text{true}$
$\nu(x_4) = \text{false}$

FIG. 7. *Graph representation of* $c'$ *(*$n = 6$*).*

Let $F_l'$ be the constraint obtained by replacing $x_i$ in $F_l$ by $s_i > t_i$, and $\neg x_i$ by $s_i < t_i$. Define $c_l = F_l' \wedge c'$, where

$$c' = \bigwedge_{i=1}^{n} ((t_{i-1} < s_i < t_{i+1}) \wedge (s_i \neq t_i)).$$

The minimum graph representation $G_{c'}^*$ of $c'$ is shown in Figure 7. Lastly, define $c = c_1 \vee \cdots \vee c_k$. Note that only the complete paths on $G_{c'}^*$ can satisfy $c$.

First, we show that $F$ is a tautology if and only if all the complete paths on $G_{c'}^*$ satisfy $c$. With each interpretation $\nu$ of $F$, associate the following complete path $w_\nu$ on $G_{c'}^*$ (see Figure 7 again):

- $w_\nu$ contains $((i,i),(i,i+1))$ if $\nu(x_i) = $ true;
- $w_\nu$ contains $((i,i),(i+1,i))$ if $\nu(x_i) = $ false.

It is not difficult to see that $\nu$ satisfies $F$ if and only if $\rho(w_\nu) \models c$. Note that $\rho(w_{\nu_{\text{true}}}) \models c$ and $\rho(w_{\nu_{\text{false}}}) \models c$ since both $\nu_{\text{true}}$ and $\nu_{\text{false}}$ satisfy $F$.

To complete the proof, we show that all the complete paths on $G_{c'}^*$ satisfy $c$ if and only if $c$ has a graph representation. For the *only if* part, suppose that all the complete paths on $G_{c'}^*$ satisfy $c$. Then, immediately from Definition 3.2, $G_{c'}^*$ is a graph representation of $c$. For the *if* part, suppose that $\rho(w) \not\models c$, where $w$ is a complete path on $G_{c'}^*$. Then, for every arc $a$ contained in $w$, there is another complete path $w_a$ (namely, $w_{\nu_{\text{true}}}$ or $w_{\nu_{\text{false}}}$; see Figure 8) that contains $a$ and $\rho(w_a) \models c$. Therefore, by Lemma 3.7, $c$ has no graph representation. $\square$

**4. A tractable subclass of graph-representable constraints.** We define a subclass CDC$^{\neq}$ of constraints such that for each constraint $c$ in CDC$^{\neq}$, a graph representation of $c$ can be constructed in $O(|c|mn)$ time.

DEFINITION 4.1. *A constraint in* CDC$^{\neq}$ *is a conjunction such that each conjunct is in the form of either* $s \neq t$ *or* $(s \geq t') \vee (s' \leq t)$*, where* $s <_S s'$ *and* $t <_T t'$*.*

CDC$^{\neq}$ can express any constraints in the form of $s_i \,\Theta\, t_j$, where $\Theta \in \{<, >, \leq, \geq, =, \neq\}$. For example,

$$(s_i \leq t_j) \equiv (s_0 \geq t_n) \vee (s_i \leq t_j),$$
$$(s_i < t_j) \equiv ((s_0 \geq t_n) \vee (s_i \leq t_j)) \wedge (s_i \neq t_j).$$

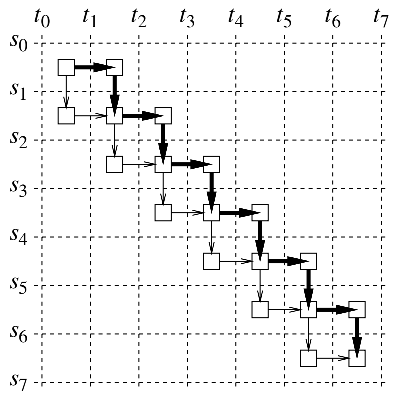

$t_0$ $t_1$ $t_2$ $t_3$ $t_4$ $t_5$ $t_6$ $t_7$
$s_0$ $s_1$ $s_2$ $s_3$ $s_4$ $s_5$ $s_6$ $s_7$

FIG. 8. $w_{\nu_{\mathrm{true}}}$ *(thick arcs) and* $w_{\nu_{\mathrm{false}}}$ *(thin arcs) in Theorem* 3.8 *(*$n = 6$*).*

Also, constraints in the form of $(s > t')\vee(s' \le t)$, $(s \ge t')\vee(s' < t)$, or $(s > t')\vee(s' < t)$ are expressible in CDC$^{\neq}$. For example,

$$(s > t') \vee (s' \le t) \equiv ((s \ge t') \vee (s' \le t)) \wedge (s \ne t').$$

As stated in section 1, constraint $(s \ge t')\vee(s' \le t)$ $(s <_S s'$ and $t <_T t')$ represents that time intervals $[s, s']$ and $[t, t']$ are disjoint. Therefore, we call the constraint a *disjointness constraint*, and we write $[s, s'] \not\pitchfork [t, t']$ to mean $(s \ge t') \vee (s' \le t)$. The class of conjunctive disjointness constraints is denoted by CDC.

Now we show that constructing a graph representation of every constraint in CDC$^{\neq}$ is tractable. By Lemma 3.3, it suffices to show that each of the two forms in Definition 4.1 has a graph representation.

LEMMA 4.2. *The following two properties hold:*

1. *Let* $c = (s_i \ne t_j)$*. A graph representation of* $c$ *is obtained by removing the arc* $((i, j), (i + 1, j + 1))$ *from* $G_{ST}$*. See Figure* 9*, for example.*
2. *Let* $c = ([s_i, s_{i'}] \not\pitchfork [t_j, t_{j'}])$*. Let* $N = \{(i'', j'') \mid i + 1 \le i'' \le i'$ *and* $j + 1 \le j'' \le j'\}$*. A graph representation of* $c$ *is obtained by removing* $N$ *and the adjacent arcs from* $G_{ST}$*. See Figure* 10*, for example.*

*Proof.* Since the first property is obvious, we consider only the case that $c = ([s_i, s_{i'}] \not\pitchfork [t_j, t_{j'}])$. Let $G$ be the graph obtained by this lemma. We show that $G$ is a graph representation of $c$. That is, $\sigma \models c$ if and only if $\rho^{-1}(\sigma)$ is a complete path on $G$.

Suppose that $\sigma \not\models c$. Then $\sigma \models (s_i < t_{j'}) \wedge (s_{i'} > t_j)$. This means that $\rho^{-1}(\sigma)$ contains some $(i'', j'')$ such that $i + 1 \le i'' \le i'$ and $j + 1 \le j'' \le j'$. Since $G$ does not contain $(i'', j'')$ by definition, $\rho^{-1}(\sigma)$ is not a complete path on $G$.

Conversely, consider a complete path $\rho^{-1}(\sigma)$ on $G_{ST}$ which is not contained in $G$. Then there is a node $(i'', j'')$ $(i + 1 \le i'' \le i'$ and $j + 1 \le j'' \le j')$ in $\rho^{-1}(\sigma)$ which is contained in $G_{ST}$ but not in $G$. Therefore, we conclude that $\sigma \models (s_i < t_{j'})\wedge(s_{i'} > t_j)$. That is, $\sigma \not\models c$. □

THEOREM 4.3. *Let* $c \in$ CDC$^{\neq}$*. A graph representation of* $c$ *can be constructed in* $O(|c|mn)$ *time. The minimum graph representation can also be constructed in* $O(|c|mn)$ *time.*

*Proof.* The proof is immediate from Lemmas 3.3, 3.6, and 4.2. □

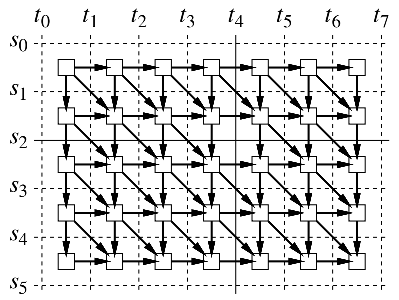

$t_0$ $t_1$ $t_2$ $t_3$ $t_4$ $t_5$ $t_6$ $t_7$
$s_0$ $s_1$ $s_2$ $s_3$ $s_4$ $s_5$

FIG. 9. *Minimum graph representation of* $s_2 \neq t_4$.

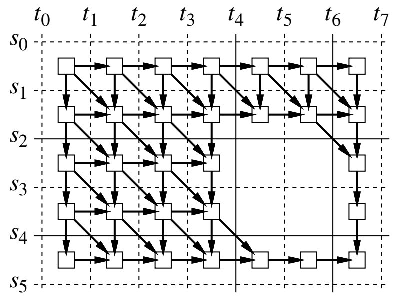

$t_0$ $t_1$ $t_2$ $t_3$ $t_4$ $t_5$ $t_6$ $t_7$
$s_0$ $s_1$ $s_2$ $s_3$ $s_4$ $s_5$

FIG. 10. *Minimum graph representation of* $[s_2, s_4] \not\cap [t_4, t_6]$.

THEOREM 4.4. *Let c be a disjunction of constraints in* $\mathrm{CDC}^{\neq}$. *The consistency of c is decidable in* $O(|c|mn)$ *time.*

*Proof.* The proof is immediate from Theorem 4.3 and Lemma 3.4. □

**5. Intractability of general disjointness constraints.** Recall the explanation of disjointness constraints in Example 1.1. Now we consider the case in which observations may contain uncertainty. For example, suppose that $a_S$ observed that $p$ or $p'$ was true during time interval $[s, s']$. Also suppose that $a_T$ observed that $p$ was false during $[t, t']$, and $p'$ was false during $[u, u']$. Then we obtain $([s, s'] \not\cap [t, t']) \vee ([s, s'] \not\cap [u, u'])$.

As stated above, observations with uncertainty bring disjunction into disjointness constraints. In this section, we show that both the consistency and the graph representability for disjointness constraints with disjunction are intractable.

**5.1. Consistency of general disjointness constraints.** We show that the consistency of general disjointness constraints is NP-complete.

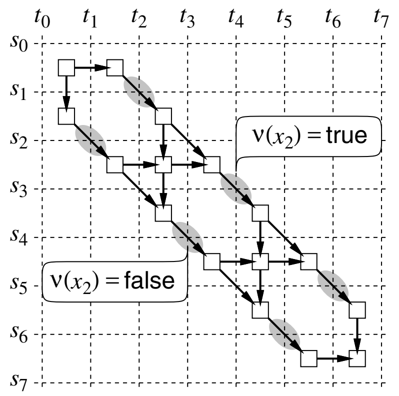
THEOREM 5.1. *Let d be a constraint in CNF with respect to disjointness constraints; i.e., d is in the form of* $d_1 \wedge \cdots \wedge d_n$, *where each* $d_i$ *is a disjunction of disjointness constraints. Then the consistency of d is NP-complete.*

*Proof.* By Theorem 2.1, the problem is in NP. To see the NP-hardness, we reduce the satisfiability of CNF logical formulas to the consistency problem.

Let $F$ be a CNF formula with $n$ variables $x_1, \ldots, x_n$. Let

$$S = \{s_0, \ldots, s_{2n+1}\}, \quad T = \{t_0, \ldots, t_{2n+1}\},$$

and let $F'$ be the constraint obtained by replacing $x_i$ in $F$ by $[s_{2i}, s_{2i+1}] \not\cap [t_{2i-1}, t_{2i}]$,



$t_0$ $t_1$ $t_2$ $t_3$ $t_4$ $t_5$ $t_6$ $t_7$
$s_0$ $s_1$ $s_2$ $s_3$ $s_4$ $s_5$ $s_6$ $s_7$
$\nu(x_2) = \text{true}$
$\nu(x_2) = \text{false}$

FIG. 11. *Graph representation of* $d'$ $(n = 3)$.

and $\neg x_i$ by $[s_{2i-1}, s_{2i}] \not\cap [t_{2i}, t_{2i+1}]$. Also define

$$d' = \bigwedge_{i=1}^{n} ([s_{2i-1}, s_{2i}] \not\cap [t_{2i-1}, t_{2i}])$$
$$\wedge \bigwedge_{j=1}^{2n-1} ([s_{j-1}, s_j] \not\cap [t_{j+1}, t_{j+2}])$$
$$\wedge \bigwedge_{j=1}^{2n-1} ([s_{j+1}, s_{j+2}] \not\cap [t_{j-1}, t_j]).$$

The minimum graph representation $G^*_{d'}$ of $d'$ is shown in Figure 11. Last, define $d = F' \wedge d'$. Note that only the complete paths on $G^*_{d'}$ can satisfy $d$.

We show that $d$ is consistent if and only if $F$ is satisfiable. With each interpretation $\nu$ of $F$, associate a complete path $w_\nu$ on $G^*_{d'}$ satisfying the following conditions (see Figure 11 again):

- $w_\nu$ contains $((2i-1, 2i), (2i, 2i+1))$ if $\nu(x_i) = \text{true}$; and
- $w_\nu$ contains $((2i, 2i-1), (2i+1, 2i))$ if $\nu(x_i) = \text{false}$.

Such $w_\nu$ always exists. On the other hand, for every complete path $w$ on $G^*_{d'}$, there is $\nu$ such that $w = w_\nu$.

Suppose that $\nu(x_i) = \text{true}$. Then $w_\nu$ contains $((2i-1, 2i), (2i, 2i+1))$, and therefore $\rho(w_\nu)(s_{2i-1}) = \rho(w_\nu)(t_{2i})$. This means that $\rho(w_\nu)$ satisfies $[s_{2i}, s_{2i+1}] \not\cap [t_{2i-1}, t_{2i}]$ but not $[s_{2i-1}, s_{2i}] \not\cap [t_{2i}, t_{2i+1}]$. Similarly, suppose that $\nu(x_i) = \text{false}$. Then $w_\nu$ contains $((2i, 2i-1), (2i+1, 2i))$, and therefore $\rho(w_\nu)(s_{2i}) = \rho(w_\nu)(t_{2i-1})$. This means that $\rho(w_\nu)$ satisfies $[s_{2i-1}, s_{2i}] \not\cap [t_{2i}, t_{2i+1}]$ but not $[s_{2i}, s_{2i+1}] \not\cap [t_{2i-1}, t_{2i}]$. Thus $\nu$ satisfies $F$ if and only if $\rho(w_\nu) \models d$. □

**5.2. Graph representability of general disjointness constraints.** We show that the graph representability of general disjointness constraints is coNP-complete.

THEOREM 5.2. *Let $d$ be a constraint in DNF with respect to disjointness constraints; i.e., $d$ is in the form of $d_1 \vee \cdots \vee d_n$, where each $d_i$ is a conjunction of disjointness constraints. Then the graph representability of $d$ is coNP-complete.*

*Proof.* By Theorem 3.8, the problem is in coNP. To see the coNP-hardness, we reduce the modified tautology problem to the graph representability problem.

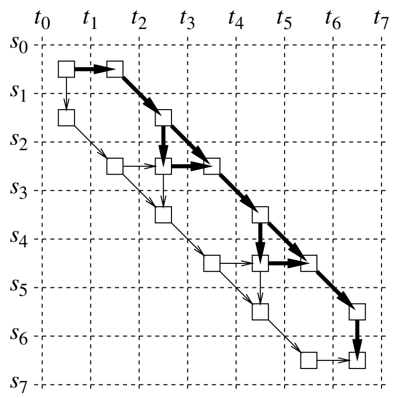


FIG. 12. $w_{\nu_{\text{true}}}$ *(thick arcs) and* $w_{\nu_{\text{false}}}$ *(thin arcs) in Theorem* 5.2 *(*$n = 3$*).*

Let $F = F_1 \vee \cdots \vee F_k$ be a DNF formula with $n$ variables $x_1, \ldots, x_n$ such that both $\nu_{\text{true}}$ and $\nu_{\text{false}}$ satisfy $F$. Let

$$S = \{s_0, \ldots, s_{2n+1}\}, \quad T = \{t_0, \ldots, t_{2n+1}\}.$$

Let $F'_l$ be the constraint obtained by replacing $x_i$ in $F_l$ by $[s_{2i}, s_{2i+1}] \not\pitchfork [t_{2i-1}, t_{2i}]$, and $\neg x_i$ by $[s_{2i-1}, s_{2i}] \not\pitchfork [t_{2i}, t_{2i+1}]$. Then define $d_l = F'_l \wedge d'$, where $d'$ is the same as the proof of Theorem 5.1. Last, define $d = d_1 \vee \cdots \vee d_k$.

In the same way as Theorem 5.1, associate a complete path $w_\nu$ on $G^*_{d'}$ with each interpretation $\nu$ of $F$. Then $\nu$ satisfies $F$ if and only if $\rho(w_\nu) \models d$. Therefore, $F$ is a tautology if and only if all the complete paths on $G^*_{d'}$ satisfy $d$. Note that $\rho(w_{\nu_{\text{true}}}) \models d$ and $\rho(w_{\nu_{\text{false}}}) \models d$ since both $\nu_{\text{true}}$ and $\nu_{\text{false}}$ satisfy $F$.

To complete the proof, we show that all the complete paths on $G^*_{d'}$ satisfy $d$ if and only if $d$ has a graph representation. For the *only if* part, suppose that all the complete paths on $G^*_{d'}$ satisfy $d$. Then, immediately from Definition 3.2, $G^*_{d'}$ is a graph representation of $d$. For the *if* part, suppose that $\rho(w) \not\models d$, where $w$ is a complete path on $G^*_{d'}$. Then, for every arc $a$ contained in $w$, there is another complete path $w_a$ (namely, $w_{\nu_{\text{true}}}$ or $w_{\nu_{\text{false}}}$; see Figure 12) that contains $a$ and $\rho(w_a) \models d$. Therefore, by Lemma 3.7, $d$ has no graph representation. $\square$

## 6. Comparison to the related works.

**6.1. Related works.** Temporal reasoning has been extensively studied mainly in the field of artificial intelligence. Allen [2] proposed the *interval algebra*, which can express the conjunction of any relation between two time intervals. Table 1 shows the 13 basic operators of the interval algebra. Every relation between two time intervals is represented by a disjunctive combination of some of these basic operators. For example, the disjointness of two intervals $I$ and $J$ is represented by $I(\mathsf{pp}^{-1}\mathsf{mm}^{-1})J$, i.e., either $I$ precedes $J$, $I$ is preceded by $J$, $I$ meets $J$, or $I$ is met by $J$. Unfortunately, many of the basic problems including the consistency (i.e., satisfiability) for the interval algebra are NP-hard [12]. Therefore, most of the researches aim to find tractable classes of temporal constraints.

One of the research directions is to weaken the expressive power of the interval algebra. Vilain and Kautz [12] proposed the *point algebra*, which can express the conjunction of any relationship between two time points. The point algebra is strictly

Table 1
*Basic interval-interval operators.*

| | | |
|---|---|---|
| I precedes J | p | III |
| J preceded by I | $\mathsf{p}^{-1}$ | JJJ |
| I meets J | m | IIII |
| J met by I | $\mathsf{m}^{-1}$ | JJJJ |
| I overlaps J | o | IIII |
| J overlapped by I | $\mathsf{o}^{-1}$ | JJJJ |
| I during J | d | III |
| J includes by I | $\mathsf{d}^{-1}$ | JJJJJJJ |
| I starts J | s | III |
| J started by I | $\mathsf{s}^{-1}$ | JJJJJJJ |
| I finishes J | f | III |
| J finished by I | $\mathsf{f}^{-1}$ | JJJJJJJ |
| I equals J | $\equiv$ | IIII |
| | | JJJJ |

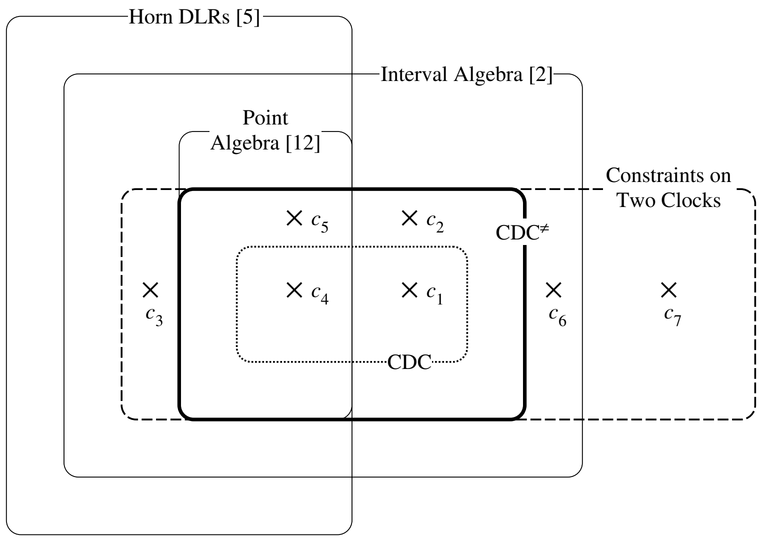
less expressive than the interval algebra, and the consistency of a given constraint $c$ is decidable in $O(|c|^2)$ time [13]. Golumbic and Shamir [3] investigated several subalgebras of the interval algebra for which the consistency is decidable in polynomial time. Nebel and Bürckert [9] proposed a subalgebra of the interval algebra called ORD-Horn, which contains the point algebra. They showed that ORD-Horn is a maximal tractable subalgebra. More precisely, the consistency of a given ORD-Horn constraint $c$ is decidable in $O(|c|^3)$ time, while the consistency for any subalgebra of the interval algebra which strictly contains ORD-Horn is NP-complete. Recently, all the tractable subalgebras of the interval algebra were discovered by Krokhin, Jeavons, and Jonsson [6].

On the other hand, some researches focus on classes incomparable to the interval algebra. Jonsson and Bäckström [5] proposed a class of temporal constraints called Horn DLRs. The class of Horn DLRs is a superclass of ORD-Horn and includes quantitative constraints (and therefore Horn DLRs are incomparable to the interval algebra). The consistency of Horn DLRs is decidable in polynomial time using a fast algorithm for the linear programming problem. Van der Meyden [11] studied the complexity of determining whether a conjunction of given constraints in the form of $s \leq t$ or $s' < t$ implies a given negation-free existentially quantified constraint.

**6.2. Comparison of the expressive power.** The relationship among classes of constraints is shown in Figure 13.

First of all, note that ORD-Horn and CDC$^{\neq}$ are incomparable. In a constraint of ORD-Horn, every relation between intervals must be expressed by a conjunction of *ORD-Horn clauses*. An ORD-Horn clause is a disjunction of at most one positive atomic constraint (i.e., $s \leq t$ or $s = t$) and an arbitrary number of negative atomic constraints (i.e., $s \neq t$). Since a disjointness constraint $c_1 = [s_1, s_2] \not\cap [t_1, t_2]$ is a disjunction of two positive atomic constraints, it is not in ORD-Horn. Also, $c_2 = (s_1 \neq t_1) \wedge ([s_1, s_2] \not\cap [t_1, t_2])$ is in CDC$^{\neq}$, but in neither CDC nor ORD-Horn. On the other hand, ORD-Horn includes constraints that are not in CDC$^{\neq}$. For example, $c_3 = (s_1 \neq t_1) \vee (s_2 \neq t_2)$ is in ORD-Horn but not in CDC$^{\neq}$. Also, ORD-Horn includes some of the constraints on more than two clocks, but CDC$^{\neq}$ never includes them.

Next, note that CDC$^{\neq}$ $\cap$ "ORD-Horn" = CDC$^{\neq}$ $\cap$ "point algebra". Any clause of a constraint in CDC$^{\neq}$ consists of one atomic constraint or two positive atomic constraints. Therefore, any clause of a constraint in CDC$^{\neq}$ $\cap$ "ORD-Horn" consists



"ORD-Horn" [9] = "Horn DLRs" ∩ "Interval Algebra"

FIG. 13. *Comparison of the expressive power of the classes of constraints.*

of one atomic constraint, and hence is expressible in the point algebra.

Class CDC and the point algebra have a nonempty intersection since it contains $c_4 = ([s_0, s_i] \not\pitchfork [t_j, t_n]) \equiv (s_i \leq t_j)$ as stated in section 4. On the other hand, $c_5 = (s_2 \neq t_4)$ is not in CDC because the minimum graph representation of $c_5$ (Figure 9) cannot be a subgraph of an intersection of graph representations of disjointness constraints (Figure 10).

Let $c_6 = ([s_0, s_2] \not\pitchfork [t_4, t_5]) \vee ([s_0, s_4] \not\pitchfork [t_6, t_7])$. Then

$$\begin{aligned} c_6 &\equiv (s_0 \geq t_5) \vee (s_2 \leq t_4) \vee (s_0 \geq t_7) \vee (s_4 \leq t_6) \\ &\equiv (s_2 \leq t_4) \vee (s_4 \leq t_6). \end{aligned}$$

This is not in CDC$^{\neq}$ because $c_6$ has no graph representation as shown in section 3. On the other hand, $c_6$ is in the interval algebra, i.e., $c_6 \equiv [s_2, s_4](\mathsf{pmodd}^{-1}\mathsf{ss}^{-1}\mathsf{ff}^{-1}\equiv)[t_4, t_6]$.

Last, $c_7 = ([s_1, s_2] \not\pitchfork [t_1, t_2]) \vee ([s_3, s_4] \not\pitchfork [t_3, t_4])$ is a constraint on two clocks but not in the interval algebra, because in the interval algebra, disjunction of relations of distinct pairs of intervals is not expressible.

After the first submission of this paper, all the maximal tractable subalgebras of the interval algebra were identified by Krokhin, Jeavons, and Jonsson [6]. CDC$^{\neq}$ is incomparable to any of the tractable subalgebras. The outline of the proof is as follows. Let $A$ be a subalgebra of the interval algebra that can express all the constraints in CDC$^{\neq}$. $A$ must contain $(\mathsf{pp}^{-1}\mathsf{mm}^{-1})$ in order to express $\not\pitchfork$. Also $A$ must contain $(\mathsf{m})$ or $(\mathsf{m}^{-1})$ in order to express adjacency of intervals (e.g., $[s_0, s_1]$ and $[s_1, s_2]$ in Example 1.1). Then, from the definitions of the discovered tractable subalgebras (Table 3 of [6]), it is immediate that $A$ is not contained in any of the tractable subalgebras.

Recently, Krokhin and Jonsson [7] also found maximal tractable subclasses of Meiri's qualitative algebra [8]. The qualitative algebra contains all the relational

Table 2
*Basic point-interval operators.*

| | | |
|---|---|---|
| p before I | `b` | `p` |
| | | `  III` |
| p starts I | `s` | `  p` |
| | | `  III` |
| p during I | `d` | `   p` |
| | | `  III` |
| p finishes I | `f` | `    p` |
| | | `  III` |
| p after I | `a` | `      p` |
| | | `  III` |

operators between points, between intervals, and between a point and an interval (see Table 2). CDC$^{\neq}$ is also incomparable with any of the known tractable subclasses of the qualitative algebra. The outline of the proof is as follows. Let $A'$ be a subclass of the qualitative algebra that can express all the constraints in CDC$^{\neq}$. First, it can be shown that $\not\cap$ is not expressible by using only point-point and point-interval operators. Therefore, $A'$ must contain $(\mathsf{pp}^{-1}\mathsf{mm}^{-1})$. Next, we can conclude that in order to express adjacency of intervals, either (1) $A'$ must contain either $(\mathsf{m})$ or $(\mathsf{m}^{-1})$; or (2) $A'$ must contain both $(\mathsf{s})$ and $(\mathsf{f})$. Then it is not difficult to see that $A'$ is not contained in any of the known tractable subclasses.

**7. Conclusions.** In this paper, we have studied temporal reasoning with respect to constraints on two concurrent sequences of events. We have introduced the notion of graph representations of constraints. If a graph representation of a given constraint $c$ can be constructed in polynomial time, then the consistency of $c$ is also decidable in polynomial time. We have proposed a subclass CDC$^{\neq}$ of constraints such that a graph representation of a constraint in CDC$^{\neq}$ can be constructed in polynomial time.

We have considered only the case in which the number of local clocks is *two*. However, this assumption is for simplicity. If the number of local clocks is an *arbitrary* constant, then the idea of graph representability is applicable and the consistency is decidable in polynomial time.

As future work, constraints on durations [10] and/or quantitative constraints on two concurrent sequences of events should be studied. First-order constraints on two concurrent sequences of events also remain to be investigated.

**Acknowledgments.** The authors would like to thank Mr. Tetsuya Murakami of Furukawa Electric Co., Ltd. for his helpful discussions. The authors are also grateful to the anonymous referees for their valuable suggestions and comments.

## REFERENCES

[1] S. Abiteboul, R. Hull, and V. Vianu, *Foundations of Databases*, Addison-Wesley, Boston, MA, 1995.

[2] J. F. Allen, *Maintaining knowledge about temporal intervals*, Comm. ACM, 26 (1983), pp. 832–843.

[3] M. C. Golumbic and R. Shamir, *Complexity and algorithms for reasoning about time: A graph-theoretic approach*, J. ACM, 40 (1993), pp. 1108–1133.

[4] H. Isozaki and H. Katsuno, *Observability-based nested belief computation for multiagent systems and its formalization*, in Proceedings of the 6th International Workshop on Agent Theories, Architectures, and Languages (ATAL'99), Lecture Notes in Artificial Intelligence 1757, Springer-Verlag, Berlin, 2000, pp. 27–41.

[5] P. Jonsson and C. Bäckström, *A unifying approach to temporal constraint reasoning*, Artificial Intelligence, 102 (1998), pp. 143–155.

[6] A. Krokhin, P. Jeavons, and P. Jonsson, *Reasoning about temporal relations: The tractable subalgebras of Allen's interval algebra*, J. ACM, 50 (2003), pp. 591–640.

[7] A. Krokhin and P. Jonsson, *Extending the point algebra into the qualitative algebra*, in Proceedings of the 9th International IEEE Symposium on Temporal Representation and Reasoning (TIME'02), 2002, pp. 28–35.

[8] I. Meiri, *Combining qualitative and quantitative constraints in temporal reasoning*, Artificial Intelligence, 87 (1996), pp. 343–385.

[9] B. Nebel and H.-J. Bürckert, *Reasoning about temporal relations: A maximal tractable subclass of Allen's interval algebra*, J. ACM, 42 (1995), pp. 43–66.

[10] A. K. Pujari, G. V. Kumari, and A. Sattar, *INDU: An interval and duration network*, in Proceedings of the Australian Joint Conference on Artificial Intelligence, Lecture Notes in Comput. Sci. 1747, Springer-Verlag, Berlin, 1999, pp. 291–303.

[11] R. van der Meyden, *The complexity of querying indefinite data about linearly ordered domains*, J. Comput. System Sci., 54 (1997), pp. 113–135.

[12] M. B. Vilain and H. A. Kautz, *Constraint propagation algorithms for temporal reasoning*, in Proceedings of the 5th National AAAI Conference on Artificial Intelligence (AAAI'86), 1986, pp. 377–382.

[13] M. B. Vilain, H. A. Kautz, and P. van Beek, *Constraint propagation algorithms for temporal reasoning: A revised report*, in Readings in Qualitative Reasoning about Physical Systems, Morgan-Kaufmann, San Mateo, CA, 1990, pp. 373–381.

SIAM J. COMPUT. © 2005 Society for Industrial and Applied Mathematics
Vol. 34, No. 3, pp. 515–525

# TOP-DOWN ANALYSIS OF PATH COMPRESSION*

RAIMUND SEIDEL† AND MICHA SHARIR‡

**Abstract.** We present a new analysis of the worst-case cost of path compression, which is an operation that is used in various well-known "union-find" algorithms. In contrast to previous analyses which are essentially based on bottom-up approaches, our method proceeds top-down, yielding recurrence relations from which the various bounds arise naturally. In particular the famous quasi-linear bound involving the inverse Ackermann function can be derived without having to introduce the Ackermann function itself.

**Key words.** union-find problem, disjoint sets data structure, path compression, inverse Ackermann function

**AMS subject classifications.** 68P05, 68W40

**DOI.** 10.1137/S0097539703439088

**1. Introduction.** Path compression is used in a number of algorithms, most notably in various very natural solutions to the so-called union-find problem. This problem is basic enough to be covered in most introductory texts on algorithms; however, the performance analysis of the solutions is more often than not at best incomplete, if not omitted altogether. Already the definition of the function $\alpha$, the interesting constituent of the time bound, as a quasi inverse of the Ackermann function is complicated.

Let us briefly recall the union-find problem. It asks for the maintenance of a partition of a finite set $X$ and a representative element $\rho(A) \in A$ for each set $A$ participating in the partition. There are two operations: For $x \in X$ the query FIND$(x)$ is to determine the representative element $\rho(A_x)$, where $A_x$ is the set in the partition that contains $x$. For two sets $A$ and $B$ in the current partition the operation UNION$(\rho(A), \rho(B))$ is to change the partition, replacing $A$ and $B$ by their union and providing the new representative element $\rho(A \cup B)$.

A very natural solution of this union-find problem is to represent the partition as a forest of rooted trees on the node set $X$, where each tree represents a set of the partition and its root is the representative element of that set. The operation FIND$(x)$ is realized by traversing the path from $x$ to the root $\rho$ of its containing tree (just follow parent pointers) and reporting $\rho$. The operation UNION$(\rho(A), \rho(B))$ is realized by making $\rho(A)$ the parent of $\rho(B)$, or vice versa, thus combining the two trees. Note that a single parent pointer per node suffices to implement such forests.

The time thus necessary for a UNION-operation is constant, whereas for a FIND-operation it is proportional to the length of the path traversed. Thus it is advantageous to keep paths short. To this end it makes sense to be judicious in the UNION-operation

*Received by the editors May 1, 2003; accepted for publication (in revised form) December 17, 2003; published electronically March 17, 2005. The research of this paper was supported by a grant from the U.S.-Israel Binational Science Foundation, by NSF grants CCR-97-32101 and CCR-00-98246, by a grant from the Israeli Academy of Sciences for a Center of Excellence in Geometric Computing at Tel Aviv University, and by the Hermann Minkowski–MINERVA Center for Geometry at Tel Aviv University.
http://www.siam.org/journals/sicomp/34-3/43908.html

†Fachrichtung Informatik, Univ. des Saarlandes, Postfach 151150, D-66041 Saarbrücken, Germany (rseidel@cs.uni-sb.de).

‡Tel Aviv University, School of Computer Science, Tel Aviv 69978, Israel, and Courant Institute of Mathematical Sciences, New York University, New York, NY 10012 (michas@post.tau.ac.il).

when deciding which of the two involved roots to make the new root. Choosing the one whose tree has the larger number of nodes is known as the *linking-by-weight* strategy; choosing the one with larger *rank* is known as the *linking-by-rank* strategy. Here "rank" is an integer associated with node $x$ that is initially 0 and that is increased by 1 whenever $x$ is made a parent to a node of equal rank.[1] In both strategies ties are broken arbitrarily.

Both strategies produce trees of at most logarithmic height. Thus a sequence of UNION- and $m$ FIND-operations takes at most $O(n+m\log n)$ time. Here $n=|X|$ and the initial partition consists of all singleton subsets of $X$. Note that at most $n-1$ UNION-operations can occur.

Another way of producing short find-paths is by so-called *path compression*: When performing a FIND-operation, make all nodes of the traversed path children of the root. This increases the running time of the FIND-operation only by a constant factor but may make the find-paths of subsequent operations shorter.

Analyzing the performance of algorithms employing path compression in combination with various linking strategies has a long history.

Fischer [2] showed a bound of $O(m\log\log n)$. Hopcroft and Ullman [5] proved an $O(m\log^* n)$ bound. Their approach was refined by Tarjan [7] to the so-called "multiple partition method" leading to a bound of $O(m\alpha_T(m,n))$, where

$$\alpha_T(m,n)=\min\{i\geq 1 \mid A(i,\lfloor m/n\rfloor)>\lceil\log_2 n\rceil\}$$

and $A$ is the "Ackermann function" defined by

$$\begin{aligned} A(1,j) &= 2^j && \text{for } j\geq 1,\\ A(i,1) &= A(i-1,2) && \text{for } i\geq 2,\\ A(i,j) &= A(i-1,A(i,j-1)) && \text{for } i,j\geq 2. \end{aligned}$$

Tarjan [8] also showed that this slightly superlinear bound was best possible for a reasonable, pointer-based model of computation. Later Fredman and Saks [3] proved the optimality of this bound also for the so-called cell probe model of computation. Kozen [6] simplified Tarjan's analysis of the upper bound. Later Tarjan [10] and also Harfst and Reingold [4] cast the existing analyses in terms of potential functions, a standard tool for the amortized analysis of algorithm performance. This type of analysis was taken into [1].

All the analyses so far have proceeded by low-level accounting and charging individual parent pointer changes to operations or nodes and bounding the grand total of those charges. In this paper we present an analysis that is based on a radically different approach. We proceed top-down and employ divide-and-conquer to derive recurrence relations from which the bounds follow. Our analysis yields rather precise bounds. For example, if $n=|X|<2^{66}$ and linking-by-rank and path compression are used, then any sequence of $m$ FIND-operations causes at most $\min\{m+4n, 2m+2n\}$ parent pointer changes.

## 2. The main lemma.

Let $\mathcal{F}$ be a forest of disjoint rooted trees on a finite node set $X$. Here we consider only paths $p$ in $\mathcal{F}$ that lead from some node $x$ to some ancestor $y$ of $x$. If $y$ is a root in $\mathcal{F}$, we refer to $p$ as a *rootpath*; otherwise we call it a *nonrootpath* and denote the parent of $y$ by $a(p)$.

[1] In other words the rank of $x$ is nothing but the height (number of edges on longest leaf-root path) of the tree rooted at $x$ *if no path compression had occurred.*

*Compressing a nonrootpath* $p$ means minimally changing $\mathcal{F}$ so that every node on $p$ has $a(p)$ as its parent. *Compressing a rootpath* $p$ means minimally changing $\mathcal{F}$ so that every node on $p$ becomes a root. We define the cost of such an operation to be the number of nodes that get a new parent; i.e., if $p$ is a nonrootpath with $d$ nodes, the cost is $d-1$, and if $p$ is a rootpath, the cost is 0. It is also convenient to allow empty paths involving no nodes. We classify them as rootpaths, and "compressing" them has cost 0.

Let $C = \left(p^{(i)}\right)_{1\le i\le M}$ be a sequence of paths along which compressions are performed starting with the initial forest $\mathcal{F}$; i.e., $\mathcal{F}_0 = \mathcal{F}$ and for $1 \le i \le M$ the path $p_i$ is in $\mathcal{F}_{i-1}$ and $\mathcal{F}_i$ is obtained from $\mathcal{F}_{i-1}$ by compressing $p_i$. Let $cost(C)$ denote the cumulative cost of $C$, i.e., the total number of times that some node gets a new parent. Finally we define the length of sequence $C$, $|C|$ for short, in a somewhat unorthodox way: We define $|C|$ to be the number of nonrootpaths in $C$.

Compared to the path compressions that arise in FIND-operations, the notion of path compression that we just defined is more general in two ways: First, we allow rootpath compressions. They make the presentation of our proofs simpler and play no other role. Second, we allow compressions of paths $p$ where $a(p)$ is not a root. This makes the union-find problem more static in the sense that the forest changes caused by UNION-operations can be ignored. This was already noted in [5] and [7] and is captured in the following lemma.

LEMMA 2.1. *Let $S$ be some sequence of* UNION- *and $m$* FIND-*operations on an initial partition of an $n$-element set $X$ into singletons. Let $T$ be the time necessary to execute $S$.*

*There is a forest $\mathcal{F}$ on $X$ and a sequence $C$ of $m$ path compressions, all involving nonrootpaths, so that $T = O(m + n + cost(C))$.*

*Proof.* (Sketch) For $\mathcal{F}$ consider the forest that is generated by performing just the UNION-operations of the sequence. The sequence of FIND-operations then defines a sequence of nonrootpaths in the corresponding forest. □

Thus we are interested in upper bounds for $cost(C)$ measured in terms of $|X|$ and $|C|$, the number of nonrootpath compressions in $C$.

Let us create a setup for divide-and-conquer. Consider a partition of the node set $X$ of $\mathcal{F}$ into a "bottom set" $X_b$ and a "top set" $X_t$. We call the pair $(X_b, X_t)$ a *dissection for* $\mathcal{F}$ iff $X_t$ is upwards closed in $\mathcal{F}$; i.e., if some node $x$ is in $X_t$, then every ancestor of $x$ in $\mathcal{F}$ must be in $X_t$ also. Note that a dissection $(X_b, X_t)$ cuts every path into two contiguous subpaths $p_b$ and $p_t$ consisting of the nodes on $p$ that are in $X_b$ and $X_t$, respectively. Either subpath may be empty. Also note that upward-closedness and hence also dissections are preserved under path compression. Let $\mathcal{F}(X_b)$ and $\mathcal{F}(X_t)$ be the subforests of $\mathcal{F}$ induced by the sets $X_b$ and $X_t$.

All our analyses hinge on the following main lemma.

LEMMA 2.2. *Let $C$ be a sequence of $|C|$ compress operations in a forest $\mathcal{F}$ with node set $X$. Let $(X_b, X_t)$ be an arbitrary dissection for $\mathcal{F}$.*

*There are sequences $C_b$ and $C_t$ of compress operations for $\mathcal{F}(X_b)$ and $\mathcal{F}(X_t)$, respectively, with $|C_b| + |C_t| \le |C|$ and*

$$cost(C) \le cost(C_b) + cost(C_t) + |X_b| + |C_t| .$$

*Proof.* Let $C = \left(p^{(i)}\right)_{1\le i\le M}$ be the sequence of paths to be compressed and let $\left(\mathcal{F}^{(i)}\right)_{0\le i\le M}$ be the resulting sequence of forests.

Let $(X_b, X_t)$ be a dissection of $\mathcal{F} = \mathcal{F}^{(0)}$ and let $\mathcal{F}_b = \mathcal{F}(X_b)$ and $\mathcal{F}_t = \mathcal{F}(X_t)$ be the induced bottom and top forests.

First, we define the postulated path sequences $C_b$ and $C_t$. Intuitively they are what one gets when recording the compression sequence $C$ executed with the nodes in $X_t$ (resp., $X_b$) hidden. More formally, define $C_b = \big(p_b^{(i)}\big)_{1\le i\le M}$ and $C_t = \big(p_t^{(i)}\big)_{1\le i\le M}$. It is easy to check inductively that for each $1 \le i \le M$ the path $p_b^{(i)}$ occurs in $\mathcal{F}^{(i-1)}(X_b)$, and compressing it produces $\mathcal{F}^{(i)}(X_b)$. Thus $C_b$ is a compression sequence for $\mathcal{F}_b$. Analogously $C_t$ is a compression sequence for $\mathcal{F}_t$.

Next we establish the required bound $|C_b|+|C_t| \le |C|$. If $p^{(i)}$ is a nonrootpath in $\mathcal{F}^{(i-1)}$, then at most one of $p_b^{(i)}$ and $p_t^{(i)}$ is a nonrootpath in its respective forest. If $p^{(i)}$ is a rootpath in $\mathcal{F}^{(i-1)}$, then both $p_b^{(i)}$ and $p_t^{(i)}$ are rootpaths. Thus $|C_b|+|C_t| \le |C|$. Note that here it is crucial that only nonrootpaths contribute to the length of a sequence.

Finally we show the postulated cost inequality. If a node from $X_t$ gets a new parent (necessarily from $X_t$) when compressing path $p^{(i)}$, then exactly the same thing happens when compressing $p_t^{(i)}$. Thus the number of those types of parent changes is given by $\mathit{cost}(C_t)$. Likewise the number of times that a node from $X_b$ gets a new parent from $X_b$ is given by $\mathit{cost}(C_b)$.

A node from $X_b$ getting a new parent which is in $X_t$ can happen only when compressing a nonrootpath $p^{(i)}$ for which $p_b^{(i)}$ is a nonempty rootpath in the bottom forest. In this case all but the topmost node on $p_b^{(i)}$ get a parent from $X_t$ for the first time, which can happen to each node in $X_b$ at most once. The topmost node gets a new parent again from $X_t$ only if $p_t^{(i)}$ is nonempty, which means $p_t^{(i)}$ is a nonrootpath (since $p^{(i)}$ is a nonrootpath). So this type of event can happen at most $|C_t|$ times. (Here it is crucial that only nonrootpaths contribute to the length of a sequence.) Since $X_t$ is upwards closed, no node in $X_t$ can get a parent from $X_b$, and thus the total number of parent changes, i.e., $\mathit{cost}(C)$, is bounded by

$$\mathit{cost}(C_b) + \mathit{cost}(C_t) + |X_b| + |C_t|\,. \qquad \square$$

**3. Arbitrary linking.** Let $f(m,n)$ denote the maximum possible cost for a path compression sequence of length $m$ in a forest of $n$ nodes. By Lemma 2.1 the value $f(m,n)$ yields an asymptotic upper bound for the running time of $m$ union-find operations in a universe of $n$ elements if path compression and *arbitrary* linking is used.

Using our main lemma it is straightforward to give a rough argument that something like $f(m,n) \le (m+n/2)\log_2 n$ must hold: For a given compression sequence $C$ of length $m$, choose a dissection $(X_b, X_t)$, where both $X_b$ and $X_t$ have size about $n/2$. Using induction, the inequality of the main lemma then yields

$$\begin{aligned}\mathit{cost}(C) &\le (|C_b| + n/4)\log_2(n/2) + (|C_t| + n/4)\log_2(n/2) + n/2 + |C_t|\\ &\le (m+n/2)\log_2(n/2) + n/2 + m\\ &\le (m+n/2)\log_2 n\,.\end{aligned}$$

This argument ignores rounding issues. It can be made precise by setting $|X_b| = \lfloor n/2\rfloor$ and $|X_t| = \lceil n/2\rceil$, replacing $\log_2()$ by $\lceil\log_2()\rceil$, and noting that $\lceil\log_2\lceil n/2\rceil\rceil \le \lceil\log_2 n\rceil - 1$ for integers $n > 1$.

For large $m$ this bound can be improved to

$$f(m,n) \le (2m+n)\log_{\lceil m/n\rceil+1} n\,,$$

which is achieved by setting $k = \lceil m/n\rceil + 1$ in the following claim.

CLAIM 3.1. *For any integer $k > 1$ we have $f(m,n) \le (m + (k-1)n)\lceil \log_k n \rceil$.*

*Proof.* The bound clearly holds if $n \le k$, since each node can get a new parent at most $n-2$ times. So assume $n > k$ and let $C$ be some sequence of $m$ compress operations in a forest with $n$ nodes. Let $(X_b, X_t)$ be a dissection with $n_t = |X_t| = \lceil n/k \rceil$ and $n_b = |X_b|$. Let $C_t$ and $C_b$ be the compression sequences asserted in the main lemma and let $m_t$ and $m_b$ be their respective sizes. Then, using induction, the inequality of the main lemma implies

$$\begin{aligned} cost(C) &\le (m_b + (k-1)n_b)\underbrace{\lceil \log_k n_b \rceil}_{\le \lceil \log_k n \rceil} + (m_t + (k-1)n_t)\underbrace{\lceil \log_k n_t \rceil}_{= \lceil \log_k n \rceil - 1} + n_b + m_t \\ &= (m_t + m_b + (k-1)(n_t + n_b))\lceil \log_k n \rceil - m_t - (k-1)n_t + n_b + m_t \\ &\le (m + (k-1)n)\lceil \log_k n \rceil , \end{aligned}$$

where the last inequality follows from the fact that $n_b \le (k-1)n_t$ by construction. ◻

A bound of this form was first proved by Tarjan and van Leeuwen in [9].

**4. Linking by rank.** For every node $x$ in a forest $\mathcal{F}$, define its rank rank$(x)$ to be the height of the subtree rooted at $x$, where the height of a one-node tree is 0. We call $\mathcal{F}$ a *rank-balanced forest*, or simply *rank forest*, if for each node $x$ in $\mathcal{F}$ the following property holds: For each $i$ with $0 \le i <$ rank$(x)$ node $x$ has at least one child $y_i$ with rank$(y_i) = i$.

Rank forests arise in union-find algorithms that employ the linking-by-rank strategy. It is easy to prove by induction that in such a forest every node of rank $k$ must be the root of a subtree of size at least $2^k$. Thus the maximum rank that occurs is at most $\lfloor \log_2 n \rfloor$, where $n$ is the number of nodes. The following inheritance lemma is important for our purposes. Its straightforward proof is left to the reader.

LEMMA 4.1. *Let $\mathcal{F}$ be a rank forest with node set $X$ and maximum rank $r$. Let $s$ be some integer, let $X_{\le s}$ be the set of nodes with rank at most $s$, and let $X_{>s}$ be the set of nodes with rank exceeding $s$. Then*

(i) *$(X_{\le s}, X_{>s})$ is a dissection;*
(ii) *$\mathcal{F}(X_{\le s})$ is a rank forest with maximum rank at most $s$;*
(iii) *$\mathcal{F}(X_{>s})$ is a rank forest with maximum rank at most $r - s - 1$;*
(iv) *$|X_{>s}| \le |X|/2^{s+1}$.*

Let $f(m,n,r)$ be the maximum cost for a sequence of path compressions of length $m$ in a rank forest with $n$ nodes and maximum rank $r$. By Lemma 2.1 $f(m,n,\lfloor \log_2 n \rfloor)$ yields an asymptotic upper bound for the running time of $m$ union-find operations in a universe of $n$ elements if path compression and linking-by-rank are used.

Before we prove the best possible bound for $f(m,n,r)$, let us first show how even very simple applications of our main lemma already yield surprisingly good bounds. We mainly want to convey some intuition here, so for the sake of clarity we will be imprecise and ignore rounding issues. The formal proofs in the second part of this section will be precise.

First of all note that trivially the bound $f(m,n,r) \le (r-1)n < rn =: P(r,n)$ holds, since every node can get a new parent at most $r-1$ times, as the new parent has larger rank than the previous parent.

Now consider a compression sequence $C$ of length $m$ in a rank forest with $n$ nodes and maximum rank $r$. Apply the main lemma to the dissection $(X_{\le s}, X_{>s})$ with $s = \log_2 r$. With this choice of $s$ the top forest contains so few nodes that the trivial

$P(r,n)$ bound already yields that the cost of the top sequence $C_t$ is at most $n$:

$$\begin{aligned}\mathit{cost}(C_t) &\le f(|C_t|, |X_{>s}|, r-s-1) \le P(r-s-1, |X_{>s}|)\\ &= (r-s-1)\cdot|X_{>s}| \le r\cdot n/2^{s+1} \le r\cdot n/2^{\log_2 r} = n\,.\end{aligned}$$

From the main lemma we now get

$$\mathit{cost}(C) \le \mathit{cost}(C_b) + \underbrace{\mathit{cost}(C_t)}_{\le n} + \underbrace{|X_b|}_{\le n} + \underbrace{|C_t|}_{\le |C|-|C_b|}\quad,$$

yielding

(*) $$\mathit{cost}(C) \le \mathit{cost}(C_b) - |C_b| + 2n + |C|\,,$$

where $C_b$ is a compression sequence in a rank forest with fewer than $n$ nodes and of maximum rank $s = \log_2 r$.

Using the trivial bound $P(n,s) = n\cdot s = n\log_2 r$ to bound $\mathit{cost}(C_b)$ in (*) then yields that $\mathit{cost}(C) \le m + 2n + n\log_2 r$.

Note that since $r \le \log_2 n$ this already implies an $O(m + n\log\log n)$ bound for union-find with path compression and linking-by-rank. But of course the natural and better thing to do is to apply the inequality (*) itself to bound $\mathit{cost}(C_b)$, which yields

$$\begin{aligned}\mathit{cost}(C) &\le (\mathit{cost}(C_{bb}) - |C_{bb}| + 2n + |C_b|) - |C_b| + 2n + |C|\\ &= \mathit{cost}(C_{bb}) - |C_{bb}| + 4n + |C|\,,\end{aligned}$$

where $C_{bb}$ is now a compression sequence in a rank forest with fewer than $n$ nodes and of maximum rank $\log_2 s = \log_2\log_2 r$. Of course inequality (*) can now be used to bound $\mathit{cost}(C_{bb})$, and this can be done again and again, yielding bounds of the form

$$\mathit{cost}(C) \le \mathit{cost}(C') - |C'| + 2jn + |C|\,,$$

where $C'$ is a compression sequence in a rank forest with fewer than $n$ nodes and of maximum rank

$$\underbrace{\log\cdots\log}_{j\text{ times}} r = \log^{(j)} r.$$

By definition, for $j = \log^* r$ we have $\log^{(j)} r \le 1$ and we are dealing with a rank forest of maximum rank 1. In such a forest any compression sequence has cost 0, and thus we get

$$\mathit{cost}(C) \le 2n\log^* r + |C|\,,$$

which already implies an $O(m+n\log^* n)$ bound for union-find with path compression and linking-by-rank.

But this is not the end of the story. We now know the bound $f(m,n,r) \le m + 2n\log^* r$. Knowing this bound, we can start over to bound the cost of $C$ and apply the main lemma to the dissection $(X_{\le s}, X_{>s})$ with $s = \log\log^* r$. Again with this choice of $s$ the top forest contains so few nodes that the bound already yields that the cost of the top sequence $C_t$ is at most $n + |C_t|$, yielding a bound analogous to (*) of the form

(**) $$\mathit{cost}(C) \le \mathit{cost}(C_b) - 2|C_b| + 2n + 2|C|\,,$$

where $C_b$ is a compression sequence in a rank forest with fewer than $n$ nodes and of maximum rank $s = \log\log^* r$. Inequality (**) can then be iterated again, yielding the bound

$$\mathit{cost}(C) \leq 2nL(r) + 2|C|\,,$$

where $L(r)$ counts how often $\log\log^*$ can be applied to $r$ until it is reduced to at most 1. This gives a new bound for $f(m,n,r)$ and the whole process can be repeated again and again.

To formalize this we need two definitions involving integer functions, the first of which is standard. Let $g : \mathbb{N} \to \mathbb{N}$ be a function with $g(n) < n$ for $n > 0$. Define the functions $g^* : \mathbb{N} \to \mathbb{N}$ and $g^\diamond : \mathbb{N} \to \mathbb{N}$ as follows:

$$g^*(r) = \begin{cases} 0 & \text{if } r \leq 1, \\ 1 + g^*(g(r)) & \text{if } r > 1, \end{cases}$$

$$g^\diamond(r) = \begin{cases} g(r) & \text{if } g(r) \leq 1, \\ 1 + g^\diamond(\lceil \log_2 g(r) \rceil) & \text{if } g(r) > 1. \end{cases}$$

Note that $g^\diamond$ is essentially $(\log \circ g)^*$.

With these definitions we can state and prove the following shifting lemma.

LEMMA 4.2. *Assume there is some $k \geq 0$ and some nondecreasing function $g : \mathbb{N} \to \mathbb{N}$ with $g(r) < r$ for $r > 0$ so that*

$$f(m,n,r) \leq km + 2ng(r)$$

*holds for all $m,n,r$. Then the following bound also holds for all $m,n,r$:*

$$f(m,n,r) \leq (k+1)m + 2ng^\diamond(r)\,.$$

*Proof.* We proceed by induction on $r$. The statement clearly holds if $g(r) \leq 1$, since in these cases we have $g^\diamond(r) = g(r) = 0$.

So assume $r$ is such that $g(r) > 1$ and let $C$ be some compression sequence of length $m$ in a rank forest $\mathcal{F}$ with $n$ nodes and maximum rank $r$. Let $s = \lceil \log_2 g(r) \rceil$ and let $(X_{\leq s}, X_{>s})$ be the dissection described in Lemma 4.1. Note that $s < r$ since $g(r) < r$. Put $n_b = |X_{\leq s}|$ and $n_t = |X_{>s}|$. Let $C_b$ and $C_t$ be the compression sequences asserted in the main lemma and put $m_b = |C_b|$ and $m_t = |C_t|$. By the main lemma we have

$$\mathit{cost}(C) \leq \mathit{cost}(C_b) + \mathit{cost}(C_t) + n_b + m_t\,.$$

The bottom forest $\mathcal{F}(X_{\leq s})$ is a rank forest of maximum rank $s < r$. Using the inductive assumption we get

$$\begin{aligned} \mathit{cost}(C_b) &\leq (k+1)m_b + 2n_b g^\diamond(s) \\ &\leq (k+1)m_b + 2n\,\underbrace{g^\diamond(\lceil \log_2 g(r) \rceil)}_{g^\diamond(r)-1} \\ &= (k+1)m_b + 2ng^\diamond(r) - 2n\,. \end{aligned}$$

The top forest $\mathcal{F}(X_{>s})$ is a rank forest of maximum rank $r - s - 1$ with

$$n_t \leq n/2^{s+1} = n/2^{1+\lceil \log_2 g(r) \rceil} \leq n/(2g(r))$$

nodes. Using the assumption of the lemma and the fact that $g$ is nondecreasing, we therefore get

$$cost(C_t) \le km_t + 2n_t g(r-s-1) \le km_t + n\,.$$

Putting things together, we get

$$\begin{aligned} cost(C) &\le \Big((k+1)m_b + 2ng^{\diamond}(r) - 2n\Big) + \Big(km_t + n\Big) + n_b + m_t \\ &\le (k+1)(m_b+m_t) + 2ng^{\diamond}(r) \\ &= (k+1)m + 2ng^{\diamond}(r)\,. \end{aligned}$$

Since this is true for any sequence $C$ of length $m$ in a rank forest with $n$ nodes and maximum rank $r$, we get the desired bound

$$f(m,n,r) \le (k+1)m + 2ng^{\diamond}(r)\,. \qquad \square$$

The shifting lemma makes it possible to take a simple but loose bound for $f(m,n,r)$ and derive from it a whole sequence of valid bounds. From these bounds we then choose a particularly good one.

COROLLARY 4.3. *Let the integer functions $J_k$ with $k \in \mathbb{N}$ be defined as*

$$J_0(r) = \lceil (r-1)/2 \rceil \quad \textit{and} \quad J_k(r) = J^{\diamond}_{k-1}(r) \textit{ for } k > 0.$$

*Then for each $k \in \mathbb{N}$ we have*

$$f(m,n,r) \le km + 2nJ_k(r)\,.$$

*Proof.* It is easy to show by induction that for each $k$ the function $J_k$ is nondecreasing and satisfies $J_k(r) < r$ for all $r > 0$.

Since a node in a rank forest of maximum rank $r$ has at most $r$ ancestors (one of which is its initial parent) it can get a new parent at most $r-1$ times. Therefore

$$f(m,n,r) \le n \cdot (r-1) \le 2nJ_0(r)$$

holds. For $k > 0$ the stated bound now follows by induction using the shifting lemma. $\square$

The $J_k$'s are very slowly growing functions even for small $k$'s. The reader may check that $J_1(r) \le 2$ and $J_2(r) \le 1$ for $r \le 65$. Since in a rank forest with $n$ nodes and of maximum rank $r$ we have $r \le \lfloor \log_2 n \rfloor$, we therefore get for $n < 2^{66}$ a bound of

$$f(m,n,r) \le \min\{m + 4n, 2m + 2n\}\,.$$

For general $m$ and $n$ we now want to choose from the infinitely many bounds provided by the $J_k$'s one that is particularly good. To this end define

$$\alpha_S(m,n) = \min\{k \in \mathbb{N} | J_k(\lfloor \log_2 n \rfloor) \le 1 + m/n\}\,.$$

Using the corollary we then get the following.

THEOREM 4.4. *For all $m, n, r$ we have $f(m,n,r) \le (\alpha_S(m,n) + 2)m + 2n$.*

Invoking Lemma 2.1 we immediately get the following.

THEOREM 4.5. *Performing a sequence of* UNION-*operations and* $m$ FIND-*operations on a set of size* $n$ *using union-by-rank and path compression requires at most* $O(n + m\alpha_S(m, n))$ *time.*

The reader may wonder how the function $\alpha_S(m, n)$ defined here relates to Tarjan's initial definition of the inverse Ackermann function $\alpha_T(m, n)$. In analogy to the formalism used in this paper, $\alpha_T$ can be defined as follows:

$$\alpha_T(m, n) = \min\{k \geq 1 \mid T_k(\lfloor \log_2 n \rfloor) < m/n\},$$

where

$$T_1(r) = \lfloor \log_2 r \rfloor \quad \text{and} \quad T_k(r) = T^{\bullet}_{k-1}(r) \text{ for } k > 1,$$

and

$$g^{\bullet}(r) = \begin{cases} 0 & \text{if } r \leq 2 \quad \text{(in contrast to 1 in the definition of } g^*\text{)}, \\ 1 + g^{\bullet}(g(r)) & \text{if } r > 2. \end{cases}$$

One can see that the differences between $\alpha_S$ and $\alpha_T$ are minor, and one can show that asymptotically they are equivalent.

## 5. Linking by weight.

Besides linking-by-rank, the method of linking-by-weight is another standard strategy in union-find algorithms. Applying our top-down approach to its analysis is a bit less straightforward than in the case of linking by rank that we just saw. The main technical issue is to create a setting where some analogue of Lemma 4.1 holds.

Let $\mathcal{F}$ be a forest, and for each node $x$ in $\mathcal{F}$ let $T_x$ be the subtree rooted at $x$, let $h_x$ be the height of $T_x$, and let $w(x)$ be some positive integer weight function on the nodes $x$ of $\mathcal{F}$. For each node $x$ define $W(x) = \sum_{y \text{ node in } T_x} w(y)$. We call $\mathcal{F}$ a *weight-balanced forest* iff for every node $x$ in $\mathcal{F}$ with parent $y$ we have $W(y) \geq 2W(x)$. We say such a forest is of type $(\mu, h, W)$ iff

(i) $w(x) \geq 2^\mu$ for every leaf $x$,

(ii) $h_x \leq h$ for each node $x$,

(iii) $\sum_{y \text{ node in } \mathcal{F}} w(y) \leq W$.

Note that any forest created by linking-by-weight with $n$ initial nodes, each of weight 1, is a weight-balanced forest of type $(0, \lfloor \log_2 n \rfloor, n)$.

LEMMA 5.1. *Let* $\mathcal{F}$ *be a weight-balanced forest with node set* $X$ *and node weight function* $w$ *and let* $\mathcal{F}$ *be of type* $(\mu, h, W)$. *Let* $s$ *be some integer, let* $X_{\leq s}$ *be the set of the nodes* $x$ *with* $h_x \leq s$, *and let* $X_{>s}$ *be the set of nodes with* $h_x > s$. *Define a new weight function*

$$w'(x) = \begin{cases} w(x) & \textit{if } x \in X_{\leq s}, \\ w(x) + \sum_{y \in X_{\leq s} \textit{ child of } x} W(y) & \textit{if } x \in X_{>s}. \end{cases}$$

*The following hold:*

(i) $(X_{\leq s}, X_{>s})$ *is a dissection.*

(ii) $\mathcal{F}(X_{\leq s})$ *with weight function* $w'$ *is a weight-balanced forest of type* $(\mu, s, W)$.

(iii) $\mathcal{F}(X_{>s})$ *with weight function* $w'$ *is a weight-balanced forest of type* $(\mu + s + 1, h - s - 1, W)$.

(iv) $|X_{\leq s}| \leq |X| < 2W/2^\mu$ *and* $|X_{>s}| < 2W/2^{\mu+s+1}$.

*Proof.* Points (i) and (ii) are trivial. Point (iii) follows from the property $W(\textit{parent}(x)) \geq 2W(x)$ and from the fact that the definition of the new weights

$w'()$ implies that for each node $x \in X_t$ the value $W'(x)$, defined with respect to $\mathcal{F}(X_{>s})$, coincides with $W(x)$ defined with respect to $\mathcal{F}$.

For point (iv) it suffices to show that the number of nodes in a weight-balanced forest $\mathcal{F}$ of type $(\mu, h, W)$ is less than $2W/2^{\mu}$. For each $i \in \mathbb{N}$ let $N_i = \{x \text{ node in } \mathcal{F} | h_x = i\}$ and let $W_i = \sum_{x \in N_i} W(x)$. Note that $W_i \leq W$ for each $i$. Since for each $x \in N_i$ we have $W(x) \geq 2^{\mu+i}$ (a consequence of the property $W(\mathit{parent}(x)) \geq 2W(x)$), we get

$$|N_i| \leq W_i/2^{\mu+i} \leq W/2^{\mu+i}.$$

Summing over all $i$ yields the desired bound on the number of nodes in $\mathcal{F}$. $\square$

Let $f(m, \mu, h, W)$ denote the maximum cost for a sequence of $m$ path compressions in a weight-balanced forest of type $(\mu, h, W)$. Again a shifting lemma holds.

Lemma 5.2. *Assume there is some $k \geq 0$ and some nondecreasing function $g : \mathbb{N} \to \mathbb{N}$ with $g(h) < h$ for $h > 0$ so that*

$$f(m, \mu, h, W) \leq km + 4(W/2^{\mu})g(h)$$

*holds for all $m, \mu, h, W$. Then also the following bound holds for all $m, \mu, h, W$:*

$$f(m, \mu, h, W) \leq (k+1)m + 4(W/2^{\mu})g^{\diamond}(h).$$

The proof proceeds in the same way as the proof of Lemma 4.2, using the dissection properties described in Lemma 5.1 instead of the ones described in Lemma 4.1, and with $h$ playing the role of $r$.

We can now conclude that

$$f(m, \mu, h, W) \leq km + 4(W/2^{\mu})J_k(h)$$

for every $k \in \mathbb{N}$.

As already noted, in forests as they arise in union-find algorithms with linking-by-weight, we have $w(x) = 1$ for each of the $n$ nodes, and hence $\mu = 0$, $W = n$, and $h \leq \log_2 n$. Thus we get the following.

Theorem 5.3. *A sequence of $m$ path compressions in a forest of $n$ nodes as it arises in union-find algorithms with linking-by-weight has cost at most*

$$(\alpha_S(m, n) + 4)m + 4n.$$

Again invoking Lemma 2.1 we immediately get the following.

Theorem 5.4. *Performing a sequence of* Union*-operations and $m$* Find*-operations on a set of size $n$ using union-by-weight and path compression requires at most $O(n + m\alpha_S(m, n))$ time.*

## 6. Open problems.

Can this top-down analysis method be made to work also for variants of path compression such as path compaction (see [9])? Most likely this will require some additional ideas, because, in contrast to compression, the compaction of rootpaths is nontrivial and can incur costs.

Another question is whether such a top-down approach be used to prove lower bounds involving the inverse Ackermann function.

## REFERENCES

[1] T. H. Corman, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms*, 2nd ed., MIT Press, Cambridge, MA, 2001.

[2] M. J. Fischer, *Efficiency of equivalence algorithms*, in Complexity of Computer Computations, R. E. Miller and J. W. Thatcher, eds., Plenum Press, New York, 1972, pp. 153–168.

[3] M. L. Fredman and M. E. Saks, *The cell probe complexity of dynamic data structures*, in Proceedings of the 26th Annual ACM Symposium on Theory of Computing, 1989, pp. 345–354.

[4] G. C. Harfst and E. M. Reingold, *A potential-based amortized analysis of the union-find data structure*, ACM SIGACT News, 31 (3) (September 2000), pp. 86–95.

[5] J. E. Hopcroft and J. D. Ullman, *Set merging algorithms*, SIAM J. Comput., 2 (1973), pp. 294–303.

[6] D. C. Kozen, *The Design and Analysis of Algorithms*, Texts Monogr. Comput. Sci., Springer, New York, 1992.

[7] R. E. Tarjan, *Efficiency of a good but not linear set union algorithm*, J. ACM, 22 (1975), pp. 215–225.

[8] R. E. Tarjan, *A class of algorithms which require nonlinear time to maintain disjoint sets*, J. Comput. System Sci., 18 (1979), pp. 110–127.

[9] R. E. Tarjan and J. van Leeuwen, *Worst-case analysis of set union algorithms*, J. ACM, 31 (1984), pp. 245–281.

[10] R. E. Tarjan, *Princeton cs423 class notes*, 1999; available online from http://www.cs.princeton.edu/courses/archive/spring99/cs423/handouts.html.

SIAM J. COMPUT.
Vol. 34, No. 3, pp. 526–552

# PSEUDO-LINE ARRANGEMENTS: DUALITY, ALGORITHMS, AND APPLICATIONS[*]

PANKAJ K. AGARWAL[†] AND MICHA SHARIR[‡]

**Abstract.** A finite collection of $x$-monotone unbounded Jordan curves in the plane is called a family of *pseudo-lines* if every pair of curves intersect in at most one point, and the two curves cross each other there. Let $L$ be such a collection of $n$ pseudo-lines, and let $P$ be a set of $m$ points in $\mathbb{R}^2$. Extending a result of Goodman [*Discrete Math.*, 32 (1980), pp. 27–35], we define a *duality* transform that maps $L$ to a set $L^*$ of points in $\mathbb{R}^2$ and $P$ to a set $P^*$ of ($x$-monotone) pseudo-lines in $\mathbb{R}^2$, so that the incidence and the "above-below" relations between the points and the pseudo-lines are preserved. We present an efficient algorithm for computing the dual arrangement $\mathcal{A}(P^*)$ under an appropriate model of computation.

We also present a dynamic data structure for reporting, in $O(m^\varepsilon + k)$ time, all $k$ points of $P$ that lie below a query arc, which is either a circular arc or a portion of the graph of a polynomial of fixed degree. This result is needed for computing the dual arrangement for certain classes of pseudo-lines arising in several applications, but is also interesting in its own right. We present a few applications of our dual arrangement algorithm, such as computing incidences between points and pseudo-lines and computing a subset of faces in a pseudo-line arrangement.

Next, we present an efficient algorithm for cutting a set of circles into arcs so that every pair of arcs intersect in at most one point, i.e., the resulting arcs constitute a collection of *pseudo-segments*. By combining this algorithm with our algorithm for computing the dual arrangement of pseudo-lines, we obtain efficient algorithms for several problems involving arrangements of circles or circular arcs, such as reporting or counting incidences between points and circles and computing a set of marked faces in arrangements of circles.

**Key words.** arrangements, duality, pseudo-lines

**AMS subject classifications.** 52C10, 52C30, 52C35, 68W40

**DOI.** 10.1137/S0097539703433900

## 1. Introduction.

The *arrangement* of a finite collection $\Gamma$ of geometric curves or surfaces in $\mathbb{R}^d$, denoted as $\mathcal{A}(\Gamma)$, is the decomposition of the space into relatively open connected cells of dimensions $0, \ldots, d$ induced by $\Gamma$, where each cell is a maximal connected set of points lying in the intersection of a fixed subset of $\Gamma$ and avoiding all other elements of $\Gamma$. Besides being interesting in their own right due to the rich geometric, combinatorial, algebraic, and topological structures that they possess, arrangements also lie at the heart of numerous geometric problems arising in a wide range of applications, including robotics, computer graphics, molecular modeling, and computer vision. The study of arrangements of lines and hyperplanes has a long, rich

[*]Received by the editors August 22, 2003; accepted for publication (in revised form) July 21, 2004; published electronically March 17, 2005. Work on this paper has been supported by a grant from the U.S.–Israeli Binational Science Foundation. Work by the first author was also supported by the NSF under grants CCR-00-86013 EIA-98-70724, EIA-99-72879, EIA-01-31905, and CCR-02-04118. Work by the second author was also supported by NSF grants CCR-97-32101 and CCR-00-98246, by a grant from the Israel Science Fund (for a Center of Excellence in Geometric Computing), and by the Hermann Minkowski–MINERVA Center for Geometry at Tel Aviv University. A preliminary version of the paper appeared in *Proceedings of the* 13*th ACM-SIAM Annual Symposium on Discrete Algorithms*, San Francisco, CA, 2002, pp. 800–809.

http://www.siam.org/journals/sicomp/34-3/43390.html

[†]Department of Computer Science, Duke University, Durham, NC 27708-0129 (pankaj@cs.duke.edu).

[‡]School of Computer Science, Tel Aviv University, Tel Aviv 69978, Israel, and Courant Institute of Mathematical Sciences, New York University, New York, NY 10012 (michas@post.tau.ac.il).

history. A summary of early work on arrangements can be found in [27, 28]. Although hyperplane arrangements already possess a rich structure, many applications (e.g., motion-planning in robotics and molecular modeling) call for a systematic study of arrangements of curved arcs in the plane and of curved surface patches in higher dimensions. There has been considerable progress in this area in the last two decades; see [10] for a review of recent results.

A collection $L$ of $n$ unbounded Jordan curves is called a family of *pseudo-lines* if every pair of curves intersect in at most one point and the two curves cross each other there. Arrangements of pseudo-lines were probably first studied by Levi [32]; see [23, 26, 28] for known results on pseudo-line arrangements. The work by Goodman and Pollack on allowable sequences [26] shows that any arrangement of pseudo-lines can be transformed into an arrangement of $x$-monotone pseudo-lines that is isomorphic to the original one. Such a transformation, however, is not efficient for the algorithms that we seek to develop, and we will thus confine our analysis to a priori given $x$-monotone pseudo-lines. For such pseudo-lines, the above-below relation between points and pseudo-lines, which will be used a lot in our analysis, is naturally defined. Many of the combinatorial results related to arrangements of lines (e.g., complexity of a single face, complexity of many faces, complexity of a level, incidences with a given set of points, etc.) hold for arrangements of pseudo-lines as well.

It has been shown [9, 11, 15, 16, 42] that various families of arcs (e.g., circular, parabolic, or graphs of polynomials of fixed degree) can be converted into a family of *pseudo-segments*, that is, subarcs with the property that each pair intersect at most once, by cutting the original arcs into a subquadratic number of pieces. Achieving this property with a quadratic number of cuts is trivial. Chan [15] has also shown that a collection of $N$ $x$-monotone pseudo-segments can be cut further into $O(N \log N)$ subarcs, each of which can be extended into an unbounded $x$-monotone curve, so that these curves constitute a family of pseudo-lines; the resulting subarcs are referred to as *extendible pseudo-segments*. One can then use the close relationship between line and pseudo-line arrangements to solve a variety of problems involving arrangements of arcs; see [2, 9, 11, 15, 42].

In this paper, we focus on algorithmic problems involving arrangements of pseudo-lines in the plane. These problems are much less studied than the corresponding combinatorial problems. Of course, one has to assume a reasonable representation of the given pseudo-lines in order to develop efficient algorithms for their manipulation. We therefore assume, for example, that the given pseudo-lines are algebraic (or semi-algebraic) curves of fixed maximum degree, and that our model of computation allows us to perform, in constant time, exact computations involving any constant number of such curves, such as computing the intersection point of a pair of pseudo-lines. However, even with these assumptions, several algorithms for line arrangements do not extend routinely to pseudo-line arrangements. A stumbling block in many of these algorithms, when we try to extend them to the case of pseudo-lines, is that they rely on some kind of a *duality* transform that maps lines to points and points to lines. Typically, one uses the duality that maps a line $\ell : y = ax + b$ to a point $\ell^* = (a, b)$ and a point $p = (\alpha, \beta)$ to the line $p^* : y = -\alpha x + \beta$ [20]. Note that $\ell$ passes above (resp., below, through) $p$ if and only if $\ell^*$ lies above (resp., below, on) $p^*$.

Burr, Grünbaum, and Sloane [14] had raised the question of whether a similar dual transform exists for pseudo-lines. Goodman [22], based on his work with Pollack on allowable sequences [24, 25, 26], defined a dual transform for (not necessarily $x$-monotone) pseudo-lines in the projective plane that preserves the incidence relation. That is, given a set $L$ of $n$ pseudo-lines and a set $P$ of $m$ points in $\mathbb{R}^2$, the transform yields a set $L^*$ of $n$ points and a set $P^*$ of $m$ pseudo-lines, so that a point $p$ of $P$ lies on a pseudo-line $\ell \in L$ if and only if the dual point $\ell^*$ lies on the dual pseudo-line $p^*$. Goodman's construction has several disadvantages from an algorithmic point of view. First, his construction is defined in the projective plane, and, consequently, it does not (and cannot, without considerable modifications) handle the above-below relation. A more significant problem, from the algorithmic point of view, is that his construction requires that for each pair of the given points, there exists an input pseudo-line passing through this pair. Although the existence of such a pseudo-line follows from the classical result of Levi [32], known as the *Levi enlargement lemma*, computing all these connecting pseudo-lines seems to be a highly nontrivial (and certainly inefficient) task.

We define a different dual transform, which may be regarded as an extension of Goodman's construction, and which overcomes the technical problems mentioned above. Suppose we have a data structure for storing the $m$ points of $P$ that supports the following operations, each in at most $f(m)$ (amortized) time: (i) determining whether any point of $P$ lies below (or above) a query pseudo-line $\ell$, and (ii) inserting a point into $P$ or deleting a point from $P$. Using this data structure as a "black box," we present a sweep-line algorithm for constructing the dual arrangement $\mathcal{A}(P^*)$ that runs in time $O((n + \chi)f(m) \log m)$, where $\chi$ is the number of vertices in $\mathcal{A}(P^*)$. We note that if $f(m)$ is small, say polylogarithmic in $m$ or of the form $O(m^\varepsilon)$, for any $\varepsilon > 0$, then this bound is nearly optimal. It is a bound of this kind that was missing so far in the algorithmic applications alluded to above.

Next, we describe a data structure for preprocessing a set $P$ of $m$ points in the plane so that all $k$ points of $P$ lying below the graph of a query fixed-degree polynomial can be reported in $O(m^\varepsilon + k)$ time.[1] It can also determine, in $O(m^\varepsilon)$ time, whether any point of $P$ lies below a query curve. A point can be inserted into or deleted from $P$ in $O(\log^2 m)$ time. Although our approach is closely based on Matoušek's algorithm [34] for reporting points that lie below a query line, a few technical difficulties must be overcome to extend this algorithm to the case of graphs of fixed-degree polynomials. A similar data structure also works for circular arcs.

Using our dual-arrangement algorithm, we show that all incidences between a set $P$ of $m$ points and a set $L$ of $n$ pseudo-lines can be reported in time $O(m^{2/3-\varepsilon}n^{2/3+2\varepsilon} + n^{1+\varepsilon} + m^{1+\varepsilon})$, provided the pseudo-lines in $L$ are extensions of bounded-degree polynomial arcs or of circular arcs, or for any other family of arcs for which a data structure with the above properties can be constructed. More precisely, we assume that all of our arcs have the same $x$-projection, and that they can be extended to pseudo-lines in some simple manner, e.g., by horizontal rays. If the arcs in $L$ intersect at $\chi$ points, the running time of the algorithm is $O(m^{2/3-\varepsilon}\chi^{1/3+\varepsilon} + n^{1+\varepsilon} + m^{1+\varepsilon})$. We also describe an algorithm, with the same running time, for computing the faces of $\mathcal{A}(L)$ that have a nonempty intersection with a set $P$ of $m$ "marking points," none of which lies on any arc of $L$. It has been shown in [18, 41] that the maximum number of incidences between $m$ points and $n$ pseudo-lines is $\Theta(m^{2/3}n^{2/3} + m + n)$, and that the same holds for the complexity of $m$ marked faces in an arrangement of $n$ pseudo-lines. Hence both of our algorithms are nearly optimal in the worst case.

Let $L$ be a family of $n$ *pseudo-circles* in the plane, which is a collection of closed Jordan curves, each pair intersecting at most twice. Recently, considerable progress has been made on the problem of splitting the curves in such a family $L$ into

[1]We follow the convention that an upper bound of the form $O(g(n, \varepsilon))$ means that for each $\varepsilon > 0$ there is a constant $c_\varepsilon$ such that the actual bound is $c_\varepsilon g(n, \varepsilon)$.

pseudo-segment arcs. This work started with the paper of Tamaki and Tokuyama [42], and has continued with recent papers of Aronov and Sharir [11], Chan [15, 16], Agarwal et al. [9], and Marcus and Tardos [33]. Since the resulting set of arcs is a collection of pseudo-segments, one can obtain bounds on the complexity of various substructures in arrangements of pseudo-circles by applying the known results for pseudo-segment arrangements. This approach has recently been used to obtain, among other results, nontrivial upper bounds on the complexity of a level in an arrangement of pseudo-circles [9, 15, 16, 33, 42], on the number of incidences between points and circles or parabolas [9, 11, 33], and on the complexity of many faces in an arrangement of circles or parabolas [2, 9, 33]. Specifically, the cited papers show that the number of incidences between $m$ points and $n$ circles or parabolas is $O(m^{2/3}n^{2/3} + m^{6/11}n^{9/11}\log^{2/11}(m^3/n) + m + n)$ (see [9, 11, 33]), and that the complexity of $m$ marked faces in an arrangement of $n$ circles or parabolas is $O(m^{2/3}n^{2/3} + m^{6/11}n^{9/11}\log^{6/11}(m^3/n) + n\log n)$ (see [2, 9, 33]). However, none of the preceding results were algorithmic. The only exception is an efficient algorithm for computing incidences between points and congruent circles. By adapting Matoušek's algorithm [35] for reporting point-line incidences, the incidences between $m$ points and $n$ unit circles can be computed in time $O(m^{2/3}n^{2/3}2^{O(\log^* n)} + (m+n)\log(m+n))$. Moreover, if there are only $\chi$ intersecting pairs among the circles, the running time reduces to $O(m^{2/3}\chi^{1/3}\log n + m\log n + n^{1+\varepsilon}(m/\chi)^{2\varepsilon/3})$.

In this paper we present an $O(n^{3/2+\varepsilon})$-time algorithm for splitting $n$ circles into $O(n^{3/2+\varepsilon})$ pseudo-segment arcs. The recent algorithms of Solan [40] and of Har-Peled and Sharir [30] can also be used or adapted for this task, but the running time of the resulting solutions would be close to $O(n^{7/4})$. Our algorithm follows the general approach of these algorithms, but it uses additional tools and a more refined analysis to obtain the bound stated above. Combining this algorithm with our new algorithms for handling arrangements of pseudo-lines, we obtain algorithms that report (or count) all incidences between $m$ points and $n$ circles in time

$$O(m^{2/3-\varepsilon}n^{2/3+2\varepsilon} + m^{6/11+3\varepsilon}n^{9/11-\varepsilon} + m^{1+\varepsilon} + n^{1+\varepsilon}). \tag{1.1}$$

One can also refine the bound so that it depends on the number $\chi$ of intersecting pairs among the given circles. The bound then becomes

$$O(m^{2/3-\varepsilon}\chi^{1/3+\varepsilon} + m^{6/11+3\varepsilon}\chi^{4/11+2\varepsilon}n^{1/11-5\varepsilon} + m^{1+\varepsilon} + n^{1+\varepsilon}), \tag{1.2}$$

where $\chi$ is the number of intersecting pairs of circles.

We also describe an algorithm for computing the faces in an arrangement of $n$ circles that contain at least one of $m$ given points, whose running time is

$$O(m^{2/3-\varepsilon}n^{2/3+2\varepsilon} + m^{6/11+3\varepsilon}n^{9/11-\varepsilon} + m^{1+\varepsilon} + n^{1+\varepsilon}).$$

Moreover, as in the case of incidences, we can also adapt the algorithm so that its running time is sensitive to $\chi$, the number of intersecting pairs of circles. If all circles have the same radius, then the running time can be improved to $O(m^{2/3-\varepsilon}\chi^{1/3+\varepsilon} + m^{1+\varepsilon} + n^{1+\varepsilon})$. These bounds are therefore close to the best known upper bounds for the corresponding combinatorial problems (as provided in [2, 9, 11, 33] and reviewed above). The previously best known algorithm for computing $m$ distinct faces in an arrangement of $n$ circles (even for congruent circles) required $O(n\sqrt{m}\log n)$ randomized expected time [38].

Finally, another application of our duality is found in [39], where it is shown that graphs drawn in the plane with their edges drawn as a collection of extendible pseudo-segments are equivalent to *pseudo-line graphs*, whose vertex set is a family $L$ of pseudo-lines, and whose edge set corresponds to some subset of the vertices of $\mathcal{A}(L)$ (where each vertex represents the edge that connects the two incident pseudo-lines). The equivalence means that two edges in the original graph cross each other if and only if the two corresponding arrangement vertices form a *diamond* (each lying between the two pseudo-lines incident to the other vertex). This property was originally observed by Tamaki and Tokuyama [43]. Many applications of this correspondence are presented in [39].

**2. Duality between points and pseudo-lines.** Let $L$ be a set of $n$ $x$-monotone pseudo-lines and $P$ a set of $m$ points in the plane. Let $W$ be a vertical strip that contains $P$ and all vertices of $\mathcal{A}(L)$. Let $\lambda$ and $\rho$ be, respectively, the left and right boundary lines of $W$. We clip the pseudo-lines of $L$ to within $W$ and thus assume that $L$ is a set of $x$-monotone arcs whose left and right endpoints lie on $\lambda$ and on $\rho$, respectively; see Figure 1 (i). Given such a family of arcs, we can apply an inverse transformation that extends each arc $\ell$ in $L$ to an unbounded $x$-monotone Jordan curve $\bar{\ell}$ by drawing leftward and rightward horizontal rays from its left and right endpoints, respectively; the resulting curves form a family of $x$-monotone pseudo-lines. We refer to $\bar{\ell}$ as the *extension* of $\ell$. An $x$-monotone Jordan arc that crosses $W$ completely splits $W$ into two regions. We will refer to each of these regions as a *pseudo-halfplane*. See Figure 1 (ii).

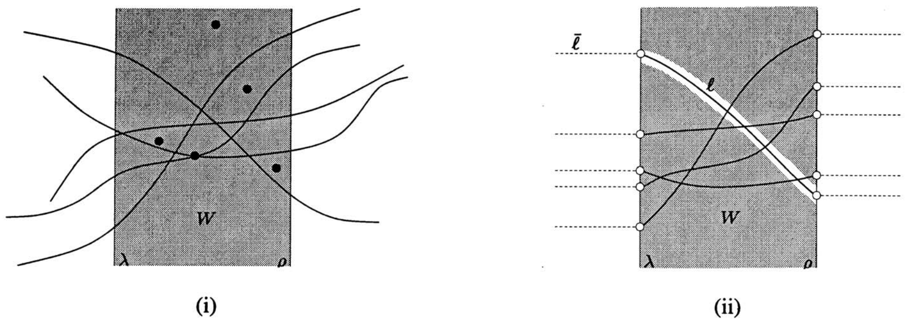


FIG. 1. (i) *A family of pseudo-lines and a set of points.* (ii) *Clipped pseudo-lines within a vertical strip $W$, extensions of the clipped arcs to Jordan curves, and two pseudo-halfplanes within $W$, bounded by a clipped pseudo-line $\ell$.*

We now present a duality transform that maps $L$ to a set $L^*$ of $n$ points and $P$ to a set $P^*$ of $m$ $x$-monotone pseudo-lines so that the incidences and the above-below relations between the points and pseudo-lines are preserved. We first describe the duality in a manner that, albeit being constructive, is not concerned with real algorithmic efficiency. We then show how to implement the construction in an efficient manner.

We sort the pseudo-lines of $L$ in increasing order of their intercepts with $\lambda$. Map each pseudo-line $\ell \in L$ to the point $\ell^* = (i_\ell, 0)$, where $i_\ell$ is the rank of the intercept $\ell \cap \lambda$ along $\lambda$. We refer to $i_\ell$ as the *index* of $\ell$. In other words, the dual points all lie on the $x$-axis and appear there from left to right in the same order as the $y$-order of the intercepts of the corresponding curves with $\lambda$. Since we are dealing with ($x$-monotone unbounded) pseudo-lines, the $y$-coordinates of the dual points, as well as the exact spacings between their $x$-coordinates, are not important. One can always move any dual point up or down (arbitrarily) or left or right (without passing over another dual point) and deform the dual pseudo-lines (that we will define shortly) accordingly, so that the incidences, the above-below relations, the $x$-monotonicity, and the pseudo-line property are all preserved. See Figure 2 for an illustration.

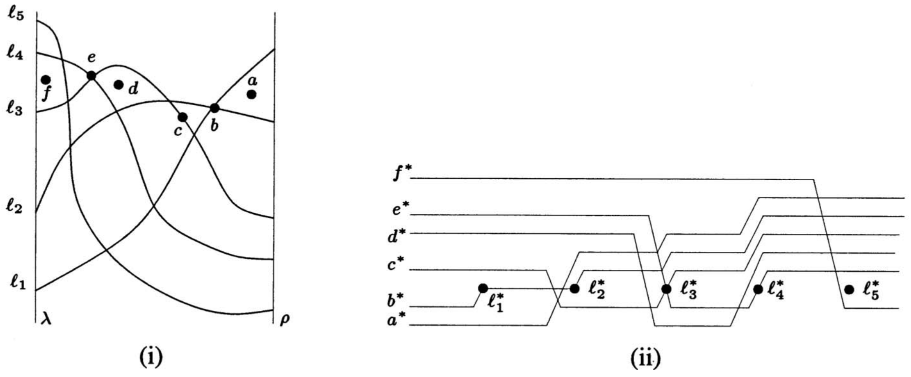


FIG. 2. *The duality transform:* (i) *The primal setting.* (ii) *The dual representation.*

Each point $p \in P$ is mapped to an $x$-monotone curve $p^*$ that is constructed so as to obey the following (necessary) rule: For each pseudo-line $\ell \in L$, if $p$ lies above (resp., below, on) $\ell$, draw $p^*$ to pass above (resp., below, through) the point $\ell^*$. This rule does not fully specify the curves $p^*$, but as we will see next, with some additional care, it yields a drawing of these curves as a collection of pseudo-lines. In order to fully specify the curves we first need to define the ordering of the dual curves $p^*$, for $p \in P$, at $x = -\infty$. Let us first assume that no pair of points of $P$ lie in the same edge or face of $\mathcal{A}(L)$. In the present course of analysis, we have no way of distinguishing between any two points that lie in the same face, and we simply regard such a pair as identical. For two points $p, q \in P$, we say that $p \prec q$ if either $x(p) < x(q)$ or $x(p) = x(q)$ and $y(p) < y(q)$. For two lines $\ell_1, \ell_2 \in L$, we say that $\ell_1 \prec \ell_2$ if the left endpoint of $\ell_1$ (on $\lambda$) lies below that of $\ell_2$.

We sort the dual curves by the order $\prec$ on the original points, and draw them at $x = -\infty$ in this order *from top to bottom*, i.e., if $p \prec q$, then $p^*$ lies above $q^*$ at $x = -\infty$. (For example, $e^*$ lies above $d^*$ at $x = -\infty$ in Figure 2 because $e \prec d$.) We draw the curves from left to right as horizontal, parallel curves until we are about to sweep past a dual point $\ell^*$. We compute the sets $P^+(\ell^*)$ (resp., $P^-(\ell^*)$, $P^\circ(\ell^*)$) consisting of those points that lie above (resp., below, on) $\ell$. This allows us to find all *inversions* enforced by $\ell^*$, namely, all pairs $(p, q)$, such that $p^*$ passes above $q^*$ to the left of $\ell^*$, but $p^*$ has to pass below or through $\ell^*$ whereas $q^*$ has to pass above or through $\ell^*$, forcing their order to reverse before we reach $\ell^*$ or at $\ell^*$ itself. We then draw the curves past $\ell^*$ so that exactly those inverted pairs cross each other just to the left of, or at $\ell^*$. To achieve this, we take all the curves in $P^+(\ell^*)$ (that have to pass above $\ell^*$) and bend them simultaneously upward, keeping them parallel to each other, so that they do not intersect among themselves, and pass above $\ell^*$. We apply a symmetric deformation downward to the curves in $P^-(\ell^*)$. Finally, we bend all curves of $P^\circ(\ell^*)$ into straight segments that pass through $\ell^*$, so that their ordering reverses after passing through $\ell^*$. (For example, $P^\circ(\ell_3^*) = \{c^*, e^*\}$, so the ordering of $c^*$ and $e^*$ reverses after they pass through $\ell_3^*$.) All these bends start at the same $x$-coordinate slightly to the left of $\ell^*$, end at the same $x$-coordinate slightly to its right (where all dual curves turn to become horizontal), and continue in this direction in parallel. In this way, it is clear that intersections in the vicinity of $\ell^*$ arise exactly between the inverted pairs. See Figure 2(ii); for simplicity we have added only those bends to the dual curves which were necessary and have omitted the others. We now prove that this procedure does indeed produce an arrangement of pseudo-lines.

LEMMA 2.1. *There do not exist two points $p, q \in P$ and two pseudo-lines $\ell_1, \ell_2 \in L$ such that*

(C1) $p \prec q$,
(C2) *$\ell_1^*$ lies to the left of $\ell_2^*$,*
(C3) *$\ell_1^*$ lies above (or on) $p^*$ and below (or on) $q^*$, and*
(C4) *$\ell_2^*$ lies below (or on) $p^*$ and above (or on) $q^*$.*

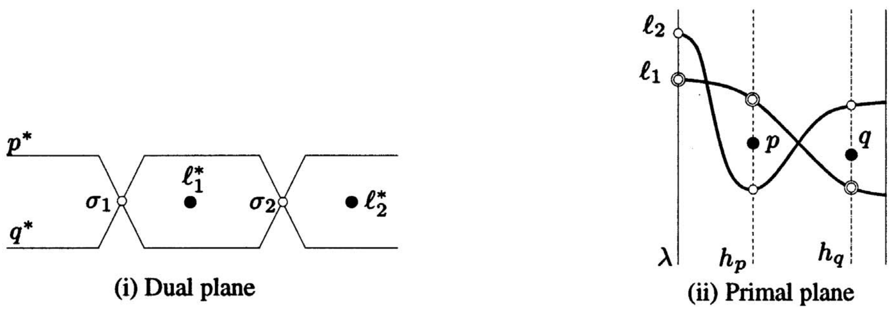


FIG. 3. *Two points $p, q$ and two curves $\ell_1, \ell_2$, satisfying* (C1)–(C4)*, in the dual and primal settings.*

*Proof.* Suppose to the contrary that there exist $p_1, p_2 \in P$ and $\ell_1, \ell_2 \in L$ that satisfy (C1)–(C4). Let $h_p$ (resp., $h_q$) be the vertical line passing through $p$ (resp., $q$). We first claim that $x(p) < x(q)$. Indeed, if $p \prec q$ and $x(p) \not< x(q)$, then $x(p) = x(q)$ and $y(p) < y(q)$. Since $\ell_2$ is $x$-monotone, it intersects the vertical line $h_p = h_q$ at a single point. Therefore $\ell_2$ cannot pass through both $p$ and $q$ if $x(p) = x(q)$. Assume that $q \notin \ell_2$. Then condition (C4) in conjunction with our construction implies that $\ell_2$ lies above $q$ but does not lie above $p$. However, both of these conditions cannot be satisfied simultaneously since $y(p) < y(q)$. The case $p \notin \ell_2$ is handled symmetrically. Hence, we can assume that $x(p) < x(q)$.

It is impossible for both $p^*$ and $q^*$ to pass through both $\ell_1^*$ and $\ell_2^*$, as this would imply that $\ell_1$ and $\ell_2$ cross twice, contradicting the fact that they are pseudo-lines. So assume that $\ell_1^* \notin q^*$; any of the symmetric cases $\ell_1^* \notin p^*$, $\ell_2^* \notin q^*$, or $\ell_2^* \notin p^*$ is handled in exactly the same manner. This in turn implies that $q \notin \ell_1$. We will consider the ordering of $\ell_1$ and $\ell_2$ along the lines $\lambda$, $h_p$, and $h_q$. Since $\ell_1^*$ lies to the left of $\ell_2^*$, $\ell_1$ lies below $\ell_2$ along $\lambda$. If $p \in \ell_1 \cap \ell_2$, then by the definition of pseudo-lines, $p$ is the only point at which $\ell_1$ and $\ell_2$ cross and $\ell_1$ lies above $\ell_2$ to the right of $h_p$. Otherwise, suppose $p \notin \ell_1$. Then by (C3) and (C4), $\ell_1$ lies above $p$ and $\ell_2$ does not lie above $p$, so $\ell_1$ lies above $\ell_2$ at $h_p$ and they cross at a point between $\lambda$ and $h_p$. See Figure 3. Once again, by (C3) and (C4) and the assumption that $q \notin \ell_1$, we can argue that $\ell_2$ lies above $\ell_1$ at the line $h_q$. Hence, $\ell_1$ and $\ell_2$ cross again between the lines $h_p$ and $h_q$. But this contradicts the assumption that $\ell_1$ and $\ell_2$ are pseudo-lines. This contradiction completes the proof of the lemma. $\square$

COROLLARY 2.2. *$P^*$ is a family of pseudo-lines.*

*Proof.* If $P^*$ were not a family of pseudo-lines, then there would exist two dual curves $p^*$ and $q^*$ that cross at least twice, say, at points $\sigma_1$ and $\sigma_2$, with $\sigma_1$ lying to the left of $\sigma_2$. Without loss of generality, assume that $p \prec q$. Then $p^*$ lies above $q^*$ at $x = -\infty$. Since $p^*$ and $q^*$ cross at $\sigma_1$, then by construction, there is an $\ell_1 \in L$ such that $p^*$ passes below or through $\ell_1^*$, $q^*$ passes above or through $\ell_1^*$, and $\ell_1^*$ lies slightly to the right of $\sigma_1$ or is equal to $\sigma_1$, but in either case it lies to the left of $\sigma_2$. The pseudo-line $p^*$ lies below $q^*$ to the right of $\sigma_1$ until their second intersection point $\sigma_2$. Again, our construction forces $p^*$ and $q^*$ to cross at $\sigma_2$ only if there is another $\ell_2 \in L$ such that $\ell_2^*$ lies to the right of $\ell_1^*$, $q^*$ passes below or through $\ell_2^*$, and $p^*$ passes above or through $\ell_2^*$. These conditions, however, yield a configuration satisfying (C1)–(C4), which contradicts Lemma 2.1. $\square$

We thus obtain the first main result of the paper.

THEOREM 2.3. *For a finite set $P$ of points and a finite set $L$ of $x$-monotone pseudo-lines in the plane, there is a duality transformation that maps $L$ into a set of points and $P$ into a set of $x$-monotone pseudo-lines so that the incidence and the above-below relations between $P$ and $L$ and between their duals are the same, i.e., $p \in P$ lies above (resp., below, on) $\ell \in L$ if and only if the pseudo-line dual to $p$ passes above (resp., below, through) the point dual to $\ell$.*

## 3. Constructing the dual arrangement.

Let $L^*$ denote the set of points dual to the pseudo-lines of $L$, and let $P^*$ denote the set of pseudo-lines dual to the points of $P$, as defined in the preceding section. We describe an efficient algorithm for computing the arrangement $\mathcal{A}(P^*)$. The output consists of a representation of $\mathcal{A}(P^*)$ as a planar map, using, e.g., the standard DCEL structure [19], which records the adjacencies between vertices and edges and between edges and faces of $\mathcal{A}(P^*)$, as well as the order of incident edges and faces about a vertex, the order of incident vertices and edges along the boundary of a face, and the order of vertices and edges along each curve of $P^*$. Moreover, the output also records, for each $\ell^* \in L^*$, the vertex, edge, or 2-face of $\mathcal{A}(P^*)$ that contains $\ell^*$.

We construct $\mathcal{A}(P^*)$ by sweeping a vertical line from left to right that stops at every point of $L^*$. We maintain the following two structures during the sweep:

(i) the ordering of curves in $P^*$ along the sweep line;
(ii) the planar map representation of $\mathcal{A}(P^*)$ restricted to the (closed) halfplane lying to the left of the sweep line.

Let $\Pi_i$ be the ordering of $P^*$ (from bottom to top) along any vertical line $\Lambda_i$ lying immediately to the right of $\ell_i^*$ (and to the left of the portion of the plane where we bend pseudo-lines of $P^*$ toward their interaction around $\ell_{i+1}^*$), let $\mathcal{A}_i$ be the arrangement of $P^*$ restricted to the (closed) halfplane lying to the left of $\Lambda_i$, and let $\mathcal{G}_i$ be the DCEL representation of $\mathcal{A}_i$. $\Pi_0$ is the ordering of $P^*$ at $x = -\infty$, as defined in section 2. $\Pi_0$ can be computed in $O(m \log m)$ time by sorting the points in $P$ in lexicographical order. $\mathcal{A}_0$, the arrangement of $P^*$ lying to the left of the "bending region" preceding $\ell_1^*$, is a planar subdivision induced by a family of $m$ horizontal segments. After having computed $\Pi_0$, $\mathcal{G}_0$ can be computed in $O(m)$ time.

When the sweep line stops at $\ell_i^*$, we compute $\Pi_i$ and $\mathcal{G}_i$ from $\Pi_{i-1}$ and $\mathcal{G}_{i-1}$. In order to expedite the computation, we maintain $\Pi_i$ in a minimum-height balanced tree $\mathcal{T}$ whose $j$th leftmost leaf stores the $j$th lowest curve in $\Pi_i$. For each node $v \in \mathcal{T}$, let $P_v \subseteq P$ denote the set of points stored at the leaves of the subtree rooted at $v$; if $v$ is a leaf, we denote the singleton point of $p$ stored at $v$ by $p_v$. At each node $v \in \mathcal{T}$, we maintain a data structure $\mathcal{D}_v = \mathcal{D}(P_v)$ that supports the following operations on $P_v$:

EMPTY($\ell$, $v$): For a pseudo-line $\ell \in L$, which of the (open) pseudo-halfplanes bounded by $\ell$, if any, contains a point of $P_v$? If neither pseudo-halfplane contains a point of $P_v$, then $P_v \subset \ell$.

INSERT$(p,\ v)$: Insert a point $p$ into $P_v$.

DELETE$(p,\ v)$: Delete a point $p$ from $P_v$.

We describe in the next section a data structure that supports these operations efficiently for certain classes of pseudo-lines. For now, assume that each of these operations can be performed in $O(f(m))$ (amortized) time.

Using this data structure, the algorithm computes $\Pi_i$ and $\mathcal{G}_i$ from $\Pi_{i-1}$ and $\mathcal{G}_{i-1}$, respectively, as follows. We visit $\mathcal{T}$ in a top-down manner. Suppose we are at a node $v \in \mathcal{T}$. We execute EMPTY$(\ell_i,\ v)$. If neither pseudo-halfplane bounded by $\ell_i$ contains a point of $P_v$, we mark $v$ by "$\circ$." If the pseudo-halfplane lying below (resp., above) $\ell_i$ does not contain a point of $P_v$ (but the complementary halfplane contains such a point), then we mark $v$ by "$+$" (resp., "$-$"). If points of $P_v$ lie on both sides of $\ell_i$, then we recursively visit the two children of $v$. Let $V^- = \langle \alpha_1, \ldots, \alpha_a \rangle$, $V^\circ = \langle \beta_1, \ldots, \beta_b \rangle$, and $V^+ = \langle \gamma_1, \ldots, \gamma_g \rangle$ be the combined sequences of leaves, sorted from left to right, lying in the subtrees rooted at the nodes marked by $-$, $\circ$, and $+$, respectively. Let $\Pi_i^- = \langle p^*_{\alpha_1}, \ldots, p^*_{\alpha_a} \rangle$, $\Pi_i^\circ = \langle p^*_{\beta_1}, \ldots, p^*_{\beta_b} \rangle$, and $\Pi_i^+ = \langle p^*_{\gamma_1}, \ldots, p^*_{\gamma_g} \rangle$. We do not compute $V^-$, $V^\circ$, $V^+$, $\Pi_i^-, \Pi_i^+, \Pi_i^\circ$ explicitly. Instead we represent them implicitly, using the roots of the subtrees where our search terminates (the nodes of $\mathcal{T}$ that were marked).

We rearrange the curves stored at the leaves of $\mathcal{T}$ so that their resulting ordering becomes $\Pi_i = \Pi_i^- \,\|\, \mathrm{rev}(\Pi_i^\circ) \| \Pi_i^+$, where $\|$ denotes concatenation and $\mathrm{rev}(\cdot)$ reverses the ordering of a list. We accomplish this in three stages, by repeatedly swapping pairs of curves stored in adjacent leaves (strictly speaking, the leaves store points of $P$, but we will not distinguish between points and their dual curves), so that:

- After the first stage all curves in $\Pi_i^-$ appear to the left of those in $\Pi_i^\circ \cup \Pi_i^+$.
- After the second stage all curves in $\Pi_i^+$ appear to the right of $\Pi_i^\circ$.
- After the third stage the ordering of curves in $\Pi_i^\circ$ is reversed.

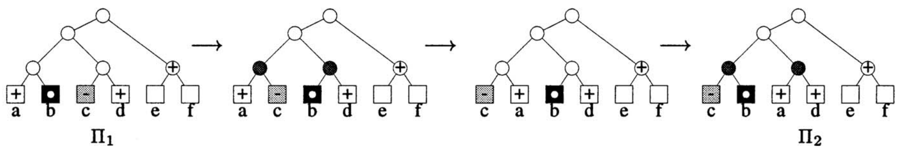


FIG. 4. *Processing $\ell_2^*$ in the dual arrangement of* Figure 2. *$\Pi_2$ is computed from $\Pi_1$ by performing the swaps $(b,c)$, $(a,c)$, and $(a,b)$. Nodes are marked $+$, $-$, and $\circ$ by the algorithm; secondary structures at the shaded interior nodes are updated.*

See Figure 4. At the end of the third stage, the curves are stored in the desired order. Whenever we swap two adjacent curves, their ordering along the sweep line gets reversed and a new vertex, along with some new faces and edges in $\mathcal{A}(P^*)$, gets created; some of the existing edges terminate at this new vertex. We update $\mathcal{A}_{i-1}, \mathcal{G}_{i-1}$ accordingly. Conceptually, while rearranging the leaves of $\mathcal{T}$, we are sweeping the vertical line from (slightly to the right of) $\ell_{i-1}^*$ to (slightly to the right of) $\ell_i^*$, swapping two curves and creating a new vertex at each step, and updating the arrangement and its planar map representation as we sweep. It is important to verify (as will be done below) that the order in which we swap the curves is topologically valid: Every pair of curves is vertically adjacent just before the curves get swapped. We call a (0-, 1-, or 2-dimensional) face of $\mathcal{A}_i$ *active* if it intersects the sweep line. At least one boundary subface of each active face lies to the right of the sweep line.

We now describe the first stage in detail. Let $\alpha_t$ be the leftmost leaf in $V^-$ that lies to the right of a leaf in $\Pi_i^\circ \cup \Pi_i^+$. Then $p^*_{\alpha_1}, \ldots, p^*_{\alpha_{t-1}}$ are in correct positions, and we move $p^*_{\alpha_t}, \ldots, p^*_{\alpha_a}$, one by one, in increasing order of their indices, to their correct positions. Suppose we have already moved $p^*_{\alpha_1}, \ldots, p^*_{\alpha_{t-1}}, p^*_{\alpha_t}, \ldots, p^*_{\alpha_{k-1}}$ to their correct positions and we now wish to bring $p^*_{\alpha_k}$ to its correct position. We repeat the following steps until $p^*_{\alpha_k}$ becomes adjacent to $p^*_{\alpha_{k-1}}$: Suppose $p^*_{\alpha_k}$ is currently stored at a leaf $w$ of $\mathcal{T}$, and $p^*_h$ is the left predecessor of $p^*_{\alpha_k}$, stored at a leaf $z$. Let $u$ be the nearest common ancestor of $w$ and $z$. We first swap the two curves stored at $w$ and $z$, that is, $w$ now stores $p^*_h$ and $z$ stores $p^*_{\alpha_k}$. Next, we delete $p^*_{\alpha_k}$ (resp., $p^*_h$) from the secondary structures $\mathcal{D}_v$ at each of the nodes $v$ on the path from $u$ to $w$ (resp., to $z$), excluding $u$, and insert $p^*_h$ (resp., $p^*_{\alpha_k}$) at $v$, using the DELETE and INSERT operations mentioned above. See Figure 5 (i).

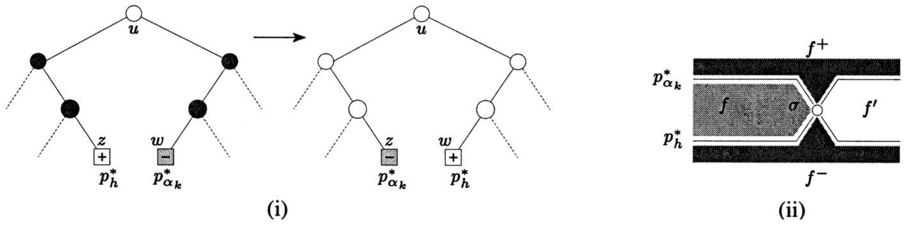


FIG. 5. (i) *Swapping two adjacent leaves $w$ and $z$ of $\mathcal{T}$.* (ii) *Swapping two curves $p^*_{\alpha_k}$ and $p^*_h$ and creating a new vertex $\sigma$, two new edges, and a 2-face; two edges and a face terminate at $\sigma$.*

Next, we create a new vertex $\sigma = p^*_{\alpha_k} \cap p^*_h$, move the sweep line over $\sigma$, and update the arrangement of $P^*$. That is, the current active edges of $p^*_h$ and $p^*_{\alpha_k}$ end at $\sigma$, with $\sigma$ being their right endpoint, and we create a new edge on each of $p^*_h$ and $p^*_{\alpha_k}$, with $\sigma$ as their left endpoint. The 2-face $f$ lying between $p^*_h$ and $p^*_{\alpha_k}$ also ends at $\sigma$, and a new 2-face $f'$ is created, with $\sigma_h$ as its leftmost endpoint and $p^*_h$ (resp., $p^*_{\alpha_k}$) as its top (resp., bottom) edge. Moreover, $p^*_h$ (resp., $p^*_{\alpha_k}$) becomes the new bottom (resp., top) edge of the 2-face $f^+$ (resp., $f^-$) that lies above $p^*_{\alpha_k}$ (resp., below $p^*_h$) to the left of $\sigma$. See Figure 5 (ii). We now update the DCEL structure to reflect these changes. Since we perform insertions and deletions in $O(\log m)$ different secondary structures, the total time spent in swapping $p^*_h$ and $p^*_{\alpha_k}$ is $O(f(m)\log m)$. The local update of $\mathcal{A}(P^*)$ and its DCEL representation take only $O(1)$ time.

After we have performed the above steps for $p^*_{\alpha_t}, \ldots, p^*_{\alpha_a}$, all curves in $\Pi_i^-$ appear to the left of $\Pi_i^\circ \cup \Pi_i^+$. Note also that all the swaps that we perform are topologically valid: They process the curves $p^*_{\alpha_t}, \ldots, p^*_{\alpha_a}$ in increasing height, effectively "bending downward" each in turn and forcing it to intersect exactly those curves that lie below it but have to pass through $\ell_i^*$ or above it, and these intersections occur in the right order. Next, we perform the same procedure on the curves in $\Pi_i^+$ that appear to the left of a curve in $\Pi_i^\circ$, in decreasing order of their indices, so that all curves in $\Pi_i^+$ appear to the right of those in $\Pi_i^\circ$. Finally, we reverse the ordering of the curves in $\Pi_i^\circ$, by once again performing swaps and updating the arrangement. Unlike the previous two stages, in which each swap created a new vertex, we create only one new vertex, namely, $\ell_i^*$, and all curves in $\Pi_i^\circ$ pass through $\ell_i^*$. Again, all the changes in $\mathcal{A}(P^*)$ that correspond to these swaps occur in a topologically valid manner.

If the algorithm performs $\mu_i$ swaps while processing $\ell_i^*$, we spend $O(\mu_i f(m)\log m)$ time in rearranging the leaves of $\mathcal{T}$ and constructing $\mathcal{G}_i$ from $\mathcal{G}_{i-1}$. If $\Pi_{i-1}$ consists of $\nu_i$ maximal contiguous subsequences such that all curves within the same subsequence lie in $\Pi_i^-$ (or $\Pi_i^+, \Pi_i^\circ$), then the algorithm marks $O(\nu_i \log m)$ nodes—$O(\log m)$ nodes for each such subsequence—and spends $O(f(m))$ time at each such node. Moreover, if any of $\Pi_i^-, \Pi_i^+, \Pi_i^\circ$ is composed of more than one contiguous subsequence in $\Pi_{i-1}$, then the curves in all subsequences, or in all but one such subsequence, are swapped, so $\nu_i \le \mu_i + 3$. Hence, the total time spent by the algorithm in processing $\ell_i^*$ is $O((\mu_i + 3)f(m)\log m)$. However, each swap introduces a new crossing between a pair of curves in $P^*$; therefore the overall running time of the algorithm is $O((n+\chi)f(m)\log m)$, where $\chi = O(m^2)$ is the number of pairs of curves in $P^*$ that cross each other.

If many curves of $P^*$ pass through a single point of $L^*$, then the number of crossing pairs could be much larger than the number of vertices in $\mathcal{A}(P^*)$. The algorithm can be modified so that $\chi$ is the number of vertices instead of the number of crossing pairs. Roughly speaking, we do not explicitly swap the curves of $\Pi_i^\circ$ in the third stage. Instead at each node $v$ of $\mathcal{T}$, we store a bit that is 0 (resp., 1) if the curves in the subtree rooted at $v$ are ordered from bottom to top (resp., top to bottom). If a node $v$ is marked $\circ$, we flip this bit at $v$. This information is needed to determine the order in which the swaps are performed. We skip the straightforward details, which are very similar to the standard sweep-line algorithm for computing the intersection points in a family of segments [12]. In summary, we have shown the following.

THEOREM 3.1. *Let $L$ be a set of $n$ $x$-monotone pseudo-lines in $\mathbb{R}^2$ and $P$ a set of $m$ points in $\mathbb{R}^2$. Suppose we have a data structure that supports each of the three operations* EMPTY, INSERT, *and* DELETE *described above, in $f(m)$ amortized time. Then we can construct $\mathcal{A}(P^*)$ (in the sense prescribed in the beginning of this section) in $O((n+\chi)f(m)\log m)$ time, where $\chi$ is the number of vertices in $\mathcal{A}(P^*)$.*

We show in the next section that if $L$ is a set of circular arcs or bounded-degree polynomial arcs (with a common $x$-projection), then $f(m) = O(m^\varepsilon)$, for any $\varepsilon > 0$, so we obtain the following corollary. The last statement in the corollary is an easy consequence of the way the algorithm is implemented.

COROLLARY 3.2. *Let $L$ be a set of $n$ circular arcs or bounded-degree polynomial arcs in $\mathbb{R}^2$ with a common $x$-projection, each pair of which intersects at most once, and let $P$ be a set of $m$ points in $\mathbb{R}^2$. Then we can construct $\mathcal{A}(P^*)$ (with respect to an extension of the arcs of $L$ into pseudo-lines) in $O((n+\chi)m^\varepsilon)$ time, where $\chi$ is the number of vertices in $\mathcal{A}(P^*)$ and $\varepsilon > 0$ is an arbitrarily small constant. Moreover, for each point $p \in P$, the above algorithm can also return, within the same overall asymptotic time bound, the set of arcs in $L$ that contain $p$, as well as the arcs that lie immediately above and below $p$.*

## 4. Pseudo-halfplane range reporting.

Let $W$ be a vertical strip, and let $\Gamma$ be a (possibly infinite) collection of $x$-monotone arcs whose endpoints lie on the left and right boundaries of $W$. Each arc $\gamma \in \Gamma$ splits $W$ into two (closed) regions. As above, we call each of these regions a *pseudo-halfplane* bounded by $\gamma$. Let $P$ be a set of $m$ points lying inside the strip $W$. We wish to preprocess $P$ into a data structure that efficiently supports the three operations described in the previous section—EMPTY, INSERT, and DELETE—with respect to arcs $\gamma \in \Gamma$. We present such a data structure for two special cases: (i) $\Gamma$ is a set of circular arcs, and (ii) $\Gamma$ is a set of (portions of the) graphs of polynomials of bounded degree.

### 4.1. Querying with circular arcs.

Let $\Gamma$ be the set of $x$-monotone circular arcs whose endpoints lie on the left and right boundaries of $W$. Let $P$ be a set of $m$ points lying inside $W$. We construct a weight-balanced binary tree $\mathcal{T}$ on the points in $P$, sorted by their $y$-coordinates [37]. For a node $v \in \mathcal{T}$, let $P_v \subseteq P$ be the set of points whose $y$-coordinates are stored at the leaves of the subtree rooted at $v$, and put $m_v = |P_v|$. We map each point $p = (x_p, y_p) \in P_v$ to the point $\bar{p} = (x_p, y_p, x_p^2 + y_p^2)$

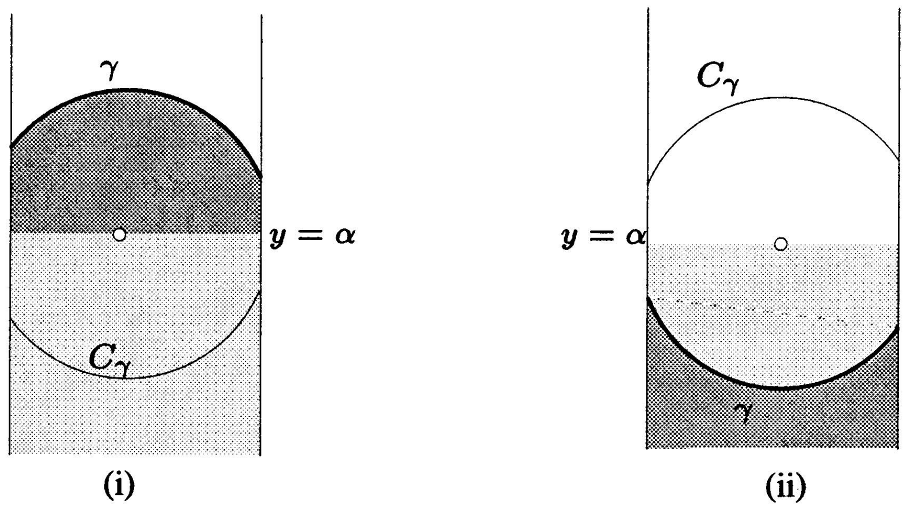


FIG. 6. *Querying with a pseudo-halfplane lying below a query arc:* (i) *The case where the arc lies in the upper semi-circle,* (ii) *The case where the arc lies in the lower semicircle.*

in $\mathbb{R}^3$. As is well known [20], if we map any circle $C$, centered at $(a, b)$ and having radius $r$, to the plane

$$\bar{C}: \; z = 2ax + 2by + (r^2 - a^2 - b^2),$$

then a point $p$ lies inside (resp., on, outside) a circle $C$ if and only if the point $\bar{p}$ lies below (resp., on, above) the plane $\bar{C}$. Let $\bar{P}_v = \{\bar{p} \mid p \in P_v\}$. We preprocess each of the sets $\bar{P}_v$ into a dynamic halfspace-emptiness data structure of size $O(m_v)$ that supports halfspace-emptiness queries (in $\mathbb{R}^3$) in time $O(m_v^\varepsilon)$ and insert/delete operations in $O(\log m_v)$ (amortized) time. See [8] for details.

Consider the halfplane-emptiness query associated with the region $g$ lying below an arc $\gamma \in \Gamma$. First, let us assume that $\gamma$ lies in the upper semicircle of its circle, denoted $C_\gamma$; let $\alpha$ denote the $y$-coordinate of the center of $C_\gamma$. We first test in $O(\log m)$ time, by checking the $y$-coordinate of the point stored in the leftmost leaf of $\mathcal{T}$, whether there exists a point of $P$ whose $y$-coordinate is at most $\alpha$. If the answer is yes, then $g \cap P \neq \emptyset$, and we terminate the query. If no such point exists, then $g \cap P \neq \emptyset$ if and only if there exists a point of $P$ inside the circle $C_\gamma$ (see Figure 6 (i)), which is equivalent to the existence of a point of $\bar{P}$ below the plane $\bar{C}_\gamma$. Using the halfspace-emptiness data structure stored at the root of $\mathcal{T}$, we determine in $O(m^\varepsilon)$ time whether any point of $P$ lies inside $C_\gamma$.

Next, assume that $\gamma$ lies on the lower semicircle of its circle $C_\gamma$. Then a point $p \in P$ lies in $g$ if and only if the $y$-coordinate of $p$ is less than $\alpha$ and $p$ lies outside the circle $C_\gamma$. We first identify in $O(\log m)$ time the $O(\log m)$ nodes $v_1, \ldots, v_s$ of $\mathcal{T}$ such that $\bigcup_{i=1}^{s} P_{v_i}$ is the set of points in $P$ whose $y$-coordinates are at most $\alpha$. For each $v_i$, we check in $O(m_{v_i}^\varepsilon)$ time whether any point of $P_{v_i}$ lies outside the circle $C_\gamma$. If the answer is yes for any $i$, then $g \cap P \neq \emptyset$; otherwise we conclude that $g \cap P = \emptyset$.

The case in which $g$ is the pseudo-halfplane lying above the arc $\gamma \in \Gamma$ can be handled in a fully symmetric manner. Using the standard partial-rebuilding technique [37], a point can be inserted into or deleted from the overall structure in $O(\log^2 m)$ amortized time. Hence, we obtain the following result (in which amortization is needed only for insertions and deletions).

THEOREM 4.1. *Let $\Gamma$ and $P$ be as above. Then each of the operations* EMPTY, INSERT*, and* DELETE *can be performed in $O(m^{\varepsilon})$ amortized time for any $\varepsilon > 0$.*

**4.2. Querying with fixed-degree polynomial arcs.** Let $\Gamma$ be the set of all arcs that are intersections with a fixed vertical strip $W$ of graphs of polynomials of degree at most $d$. We describe a dynamic data structure that determines whether any point of $P$ lies below an arc in $\Gamma$. The case of points lying above an arc is handled in a fully symmetric manner. The data structure is similar to the one proposed by Matoušek for answering halfspace-reporting queries [34]. However, we need to adapt his structure, using the results by Agarwal, Efrat, and Sharir [4] and Agarwal and Matoušek [7], to extend it to pseudo-halfplanes bounded by polynomial arcs.

We call an arc $\gamma \in \Gamma$ *$k$-shallow* if at most $k$ points of $P$ lie below $\gamma$. We call a simply connected region with at most four edges a *pseudo-trapezoid* if its top and bottom edges are portions of arcs in $\Gamma$ and its left and right edges are vertical segments. We allow pseudo-trapezoids to be degenerate, i.e., they can be unbounded or lower dimensional. A 1-dimensional pseudo-trapezoid is a portion of a curve in $\Gamma$ or a vertical segment, and a 0-dimensional pseudo-trapezoid is a point. An *elementary partition* of $P$ is a family $\Pi = \{(P_1, \triangle_1), \ldots, (P_u, \triangle_u)\}$, where $P_1, \ldots, P_u$ form a partition of $P$, and, for each $i \le u$, $\triangle_i$ is a pseudo-trapezoid, and $P_i \subseteq \triangle_i$. The pseudo-trapezoids $\Delta_i$ may intersect each other, and $\Delta_i$ may contain a point of $P_j$ for some $j \ne i$. We say that an arc $\gamma \in \Gamma$ *crosses* a cell $\triangle_i$ of $\Pi$ if $\gamma \cap \triangle_i \ne \emptyset$ and $\triangle_i \not\subseteq \gamma$. Let $\chi(\Pi, \gamma)$ be the number of cells of $\Pi$ that $\gamma$ crosses, and for any subset $Q \subseteq \Gamma$, let $\chi(\Pi, Q) = \max_{\gamma \in Q} \chi(\Pi, \gamma)$.

LEMMA 4.2. *Let $P$ be a set of $m$ points in $\mathbb{R}^2$, and let $r > 1$ be a parameter. There exists a set $Q \subseteq \Gamma$ of $O(r^{\lceil d/2 \rceil})$ $(2m/r)$-shallow curves (with respect to $P$) such that for any elementary partition $\Pi$ in which each class has at least $\lceil m/r \rceil$ points, $\chi(\Pi, \Gamma) \le (d+2)\chi(\Pi, Q)$.*

*Proof*. Let $\varphi : \mathbb{R} \to \mathbb{R}^d$ be the map $\varphi(x) = (x, x^2, \ldots, x^d)$. Then any univariate polynomial $\gamma : y = a_0 + a_1 x + \cdots + a_d x^d$ of degree $d$ maps to a $d$-variate linear function $h_\gamma : x_{d+1} = a_0 + a_1 x_1 + \cdots + a_d x_d$, in the sense that $\gamma(x) = h_\gamma(\varphi(x))$ for all $x \in \mathbb{R}$. Conversely, for any $d$-variate linear function $h : x_{d+1} = \alpha_0 + \sum_{i=1}^{d} \alpha_i x_i$, let $\gamma_h$ be the polynomial $y = \sum_{i=0}^{d} \alpha_i x^i$. Let $P^* = \{(\varphi(x), y) \mid (x, y) \in P\} \subset \mathbb{R}^{d+1}$. The notion of $k$-shallowness is extended naturally to hyperplanes in $\mathbb{R}^{d+1}$: Such a hyperplane $h$ is *$k$-shallow* with respect to $P^*$ if at most $k$ points of $P^*$ lie below $h$.

Let $H$ be a finite set of hyperplanes in $\mathbb{R}^{d+1}$. The *zone* of a hyperplane $h \notin H$ with respect to $H$, denoted as $\mathcal{Z}(H, h)$, is (the closure of) the set of points that can be reached from $h$ without intersecting any hyperplane in $H$ (it is the closure of the union of all cells crossed by $h$). We call $H$ a *$t$-guarding set* of $h$ (with respect to $P^*$) if $|\mathcal{Z}(H, h) \cap P^*| < t$. Matoušek [34, Lemma 3.3] has shown that, given a set $P^*$ of $m$ points in $\mathbb{R}^{d+1}$ and a parameter $r > 1$, there exists a family $H$ of $O(r^{\lceil d/2 \rceil})$ $(2m/r)$-shallow hyperplanes in $\mathbb{R}^{d+1}$ such that for any $(m/r)$-shallow hyperplane in $\mathbb{R}^{d+1}$, there is an $(m/r)$-guarding set $G \subseteq H$ of size at most $d+2$.

We claim that $Q = \{\gamma_h \mid h \in H\}$ is the desired set of polynomial curves. Indeed, a point $p$ lies below a curve $\gamma \in \Gamma$ if and only if $\varphi(p)$ lies below the hyperplane $h_\gamma$, so $Q$ is a set of $(2m/r)$-shallow curves. Let $\Pi = \{(P_1, \triangle_1), \ldots, (P_u, \triangle_u)\}$ be an elementary partition such that $|P_i| \ge \lceil n/r \rceil$, for any $1 \le i \le u$, and let $\eta \in \Gamma$ be a polynomial curve. If $\eta \in Q$, then obviously $\chi(\Pi, \eta) \le \chi(\Pi, Q)$. Otherwise, let $G_\eta \subseteq G$ be the guarding set of size at most $d+2$ for the hyperplane $h_\eta$. Suppose $\eta$ crosses a cell $\triangle_i$ of $\Pi$. Then $h_\eta$ crosses $\varphi(\triangle_i)$. If $\triangle_i$ is not crossed by any curve of $Q_\eta = \{\gamma_h \mid h \in G_\eta\}$, then $\varphi(\triangle_i)$ does not intersect any hyperplane of $G_\eta$ and thus lies in a single cell of $\mathcal{A}(G_\eta)$. On the other hand, $\eta$ crosses $\triangle_i$, so $h_\eta$ crosses $\varphi(\triangle_i)$, thereby implying that $\mathcal{Z}(G_\eta, h_\eta) \supset \varphi(\triangle_i) \supset \varphi(P_i)$. However, $G_\eta$ is an $(m/r)$-guarding set of $h_\eta$, so $|\mathcal{Z}(G_\eta, h_\eta) \cap \varphi(P)| < m/r$, which contradicts the fact that $\varphi(P_i) \subset \mathcal{Z}(G_\eta, h_\eta)$, as $|P_i| \geq m/r$. Hence, $\triangle_i$ is crossed by one of the curves in $Q_\eta$. Since $|Q_\eta| \leq d+2$, the lemma follows. □

The next lemma follows from a result of Matoušek [34, Lemma 3.2]. The only difference is that we use the result by Agarwal, Efrat, and Sharir [4] on "shallow-cuttings" for algebraic arcs. We omit the proof, which can be trivially adapted from the proof in the original paper.

Lemma 4.3. *Let $P$ and $\Gamma$ be as defined above, let $Q \subset \Gamma$ be a set of $r^{O(1)}$ $(2m/r)$-shallow curves, and let $r$ be a parameter. Then there exists an elementary partition $\Pi = \{(P_1, \triangle_1), \ldots, (P_u, \triangle_u)\}$ of $P$ such that $m/r \leq |P_i| \leq 2m/r$, for each $i$ (so $u \leq r$), and $\chi(\Pi, Q) = O(\log r)$. If $r$ is a constant, then $\Pi$ can be computed in $O(m)$ time.*

The following is an immediate corollary of Lemmas 4.2 and 4.3.

Lemma 4.4. *Let $P$ and $\Gamma$ be as defined above, and let $r$ be a parameter. Then there exists an elementary partition $\Pi = \{(P_1, \triangle_1), \ldots, (P_u, \triangle_u)\}$ of $P$ such that $m/r \leq |P_i| \leq 2m/r$, for each $i$ (so $u \leq r$), and $\chi(\Pi, \Gamma) = O(\log r)$. If $r$ is a constant, $\Pi$ can be computed in $O(m)$ time.*

As in [7, 34], choosing $r$ to be a sufficiently large constant and applying Lemma 4.4 recursively, we can construct, in $O(m \log m)$ time, a "partition tree" of size $O(m)$. This tree can be used for answering pseudo-halfplane-emptiness queries in $O(m^\varepsilon)$ time; see [34] for details. A point can be inserted or deleted in $O(\log^2 m)$ amortized time using the standard partial-rebuilding technique [37]. Hence, we obtain the following result.

Theorem 4.5. *Let $P$ be a set of $m$ points in some vertical strip $W$, and let $\Gamma$ be a set of fixed-degree polynomial arcs clipped to $W$. Then each of the operations* Empty, Insert, *and* Delete *can be performed in $O(m^\varepsilon)$ amortized time.*

*Remark* 4.6. As in [34], by combining the above data structure with the data structure in [7], we can report, in $O(m^\varepsilon + k)$ time, all $k$ points lying in a pseudo-halfplane bounded by a query arc in $\Gamma$. The asymptotic running time of the insertion and deletion operations remains the same.

**5. Incidences in pseudo-line arrangements.** Let $P$ be a set of $m$ points in a fixed vertical strip $W \subseteq \mathbb{R}^2$, let $L$ be a set of $n$ pseudo-lines in $\mathbb{R}^2$ that are extensions of circular or fixed-degree polynomial arcs that traverse $W$, and let $\mathcal{I}(L, P)$ denote the set of pairs $(\ell, p) \in L \times P$ such that $p$ lies on $\ell$. We wish to report $\mathcal{I}(L, P)$, compute $|\mathcal{I}(L, P)|$, or just determine whether $\mathcal{I}(L, P)$ is nonempty. The latter problem is an extension of *Hopcroft's problem* to the case of pseudo-lines. For simplicity, we focus on the first subproblem. We follow the same approach as in [6, 17]. Corollary 3.2, combined with the analysis in section 4, implies that $\mathcal{I}(L, P)$ can be computed in $O((m^2 + n)m^\varepsilon)$ time. By partitioning $P$ into $\lceil m/\sqrt{n}\ \rceil$ subsets $P_1, \ldots, P_s$, each of size at most $\sqrt{n}$, and computing $\mathcal{I}(L, P_i)$ for each subset separately, $\mathcal{I}(L, P)$ can be reported in $O(mn^{1/2+\varepsilon} + n^{1+\varepsilon})$ time, which is near optimal for $m \leq \sqrt{n}$. We now describe an algorithm that is efficient for all values of $m$ and $n$.

*Cuttings.* We first need to introduce the notion of $(1/r)$-*cuttings*. Since we will be using $(1/r)$-cuttings in the subsequent sections as well, we define them in a more general setting than needed here. Let $H$ be a set of $m$ hyperplanes in $\mathbb{R}^d$, $1 \leq r \leq m$ a parameter, and $\Delta$ a simplex. A simplicial subdivision $\Xi$ of $\Delta$ is called a $(1/r)$-*cutting* of $H$ (with respect to, or within, $\Delta$) if at most $m/r$ hyperplanes of $H$ cross any simplex of $\Xi$. We will use Chazelle's *hierarchical cuttings* [17] to construct a $(1/r)$-cutting $\Xi$ of $H$. In this approach, one chooses a sufficiently large constant $r_0$ and sets $\nu = \lceil \log_{r_0} r \rceil$. One then constructs a sequence of cuttings $\Xi_0, \Xi_1, \ldots, \Xi_\nu = \Xi$, where $\Xi_i$ is a $(1/r_0^i)$-cutting of $H$ within $\Delta$. The initial cutting $\Xi_0$ is simply $\Delta$ itself. The cutting $\Xi_i$ is obtained from $\Xi_{i-1}$ by computing, for each $\tau \in \Xi_{i-1}$, a $(1/r_0)$-cutting $\Xi_i^\tau$ of $H_\tau$ within $\tau$. It is shown in [17] that this can be done so that $|\Xi_i| \le c r_0^{di}$ for each $i$ and for some common constant $c > 0$ that depends only on $d$. Hence, $|\Xi| = O(r^d)$. If we are also given a set $P$ of $n$ points, then we can guarantee, by further partitioning simplices if needed, that each simplex in $\Xi_i$ contains at most $n/r_0^{di}$ points of $P$. The reasons for using hierarchical cuttings will become clearer in the later sections. One such reason is that it facilitates an efficient construction of $(1/r)$-cuttings in optimal $O(nr^{d-1})$ time. Chazelle's technique is presented only for the case of hyperplanes, but it admits several extensions. In the planar case, for example, it also applies to many other families of curves. In particular, we can compute in $O(mr)$ time a $(1/r)$-cutting of size $O(r^2)$ for a family $\Gamma$ of $m$ pseudo-lines (or circular arcs), in an appropriate model of computation. The only technical difference is that, instead of simplices (i.e., triangles), one needs to use vertical pseudo-trapezoids (see, e.g., [1] for details). Finally, the cutting produced by Chazelle's method can be fine-tuned, so that its size depends on the actual complexity of the arrangement of the given hyperplanes or curves. For example, in the planar case, if there are $\chi$ intersecting pairs of curves in $\Gamma$, then the size of the cutting is $O(r^{1+\varepsilon} + \chi r^2/n^2)$, for any $\varepsilon > 0$, and it can be computed in time $O(n^{1+\varepsilon} + \chi r/n)$.

Returning to the computation of $\mathcal{I}(L,P)$, we choose a parameter $r < n$ and construct a $(1/r)$-cutting $\Xi$ of $L$ of size $O(r^2)$. For a cell $\tau \in \Xi$, let $L_\tau \subseteq L$ be the set of pseudo-lines that intersect the interior of $\tau$, and let $P_\tau \subseteq P$ be the set of points that either lie in the interior of $\tau$ or lie on (the relative interior of) an edge of $\tau$. Set $n_\tau = |L_\tau|$ and $m_\tau = |P_\tau|$. Then $\sum_\tau m_\tau \le 2m$ and $n_\tau \le n/r$. At most one pseudo-line $\ell_e$ of $L$ can contain an edge $e$ of $\Xi$. If there is such a pseudo-line, we report all incidences between $e$ and the points that lie on $e$, over all edges $e$, in a total time of $O(r^2 + m)$. We compute $\mathcal{I}(L_\tau, P_\tau)$ in time $O(m_\tau n_\tau^{1/2+\varepsilon} + n_\tau^{1+\varepsilon})$ using the algorithm outlined above and repeat this for each cell $\tau$ of $\Xi$. The remaining incidences, between the pseudo-lines of $L$ and those points of $P$ that lie at the vertices of $\Xi$, are reported as follows: If an input point $p$ lies on a vertex of a cell $\tau \in \Xi$, we report in $O(n/r)$ time, using brute force, all curves of $L_\tau$ that pass through $p$. Since there are $O(r^2)$ cells in $\Xi$, the total time spent in this step is $O(nr)$. Hence, the overall running time is

$$\sum_\tau O\left(m_\tau (n/r)^{1/2+\varepsilon} + (n/r)^{1+\varepsilon}\right) + O(nr) = O\left(m(n/r)^{1/2+\varepsilon} + n^{1+\varepsilon} r^{1-\varepsilon}\right).$$

Choosing $r = \lceil m^{2/3}/n^{1/3} \rceil$ and replacing $\varepsilon$ by an appropriate constant multiple of $\varepsilon$, we obtain the following theorem.

THEOREM 5.1. *The incidences between $m$ points in a vertical strip $W$ and $n$ pseudo-lines that are extensions of circular or fixed-degree polynomial arcs that traverse $W$ can be detected, counted, or reported in time $O(m^{2/3-\varepsilon} n^{2/3+2\varepsilon} + m^{1+\varepsilon} + n^{1+\varepsilon})$ for any $\varepsilon > 0$.*

If $\mathcal{A}(L)$ has $\chi$ vertices, then, using a $(1/r)$-cutting whose size depends on $\chi$ (see above) and choosing $r = \lceil m^{2/3} n/\chi^{2/3} \rceil$, we obtain the following theorem.

THEOREM 5.2. *The incidences between $m$ points in a vertical strip $W$ and $n$ pseudo-lines with $\chi$ crossing pairs that are extensions of circular or fixed-degree polyno-*

*mial arcs that traverse W can be detected, counted, or reported in time* $O(m^{2/3-\varepsilon}\chi^{1/3+\varepsilon}+m^{1+\varepsilon}+n^{1+\varepsilon})$ *for any* $\varepsilon > 0$.

**6. Many faces in pseudo-line arrangements.** Let $L$ be a set of $n$ pseudo-lines in $\mathbb{R}^2$ that are extensions of circular or polynomial arcs, and let $P$ be a set of $m$ points in the plane, none lying on any pseudo-line of $L$. Let $\mathcal{F}(L,P)$ denote the set of faces in $\mathcal{A}(L)$ that contain at least one point of $P$. We can compute $\mathcal{F}(L,P)$ by following an approach similar to the one for computing $\mathcal{I}(L,P)$. We first describe an $O((m^2+n)n^{\varepsilon})$ algorithm for computing $\mathcal{F}(L,P)$.

We compute the arrangement $\mathcal{A}(P^*)$ of the pseudo-lines dual to the points of $P$ (with respect to $L$) and then construct its vertical decomposition $\mathcal{A}^{||}(P^*)$. For each face $\varphi \in \mathcal{A}^{||}(P^*)$, we compute the subset $L_\varphi \subseteq L$ of pseudo-lines whose dual points lie inside $\varphi$. This step takes $O((m^2+n)n^{\varepsilon})$ time. Next, we compute an Eulerian tour $\Pi$ of the planar graph dual to $\mathcal{A}^{||}(P^*)$ so that each face of $\mathcal{A}^{||}(P^*)$ is visited $O(1)$ times; see [3]. If an edge $e$ of $\Pi$ crosses two adjacent faces of $\mathcal{A}(P^*)$, we set $\pi(e)$ to be the point of $P$ whose dual pseudo-line separates these two faces (it is the dual of the curve of $P^*$ that $e$ crosses). Otherwise, $e$ connects two faces of $\mathcal{A}^{||}(P^*)$ separated by a vertical line, and we set $\pi(e) = \emptyset$. Next, we construct a minimum-height binary tree $\mathcal{T}$ on $\Pi$. Each leaf of $\mathcal{T}$ is associated with a node of $\Pi$, i.e., with a face of $\mathcal{A}^{||}(P^*)$, and each internal node $v$ of $\mathcal{T}$ is associated with the subpath $\Pi_v$ of $\Pi$ that connects the leaves of the subtree rooted at $v$. For each node $v$ of $\mathcal{T}$, we set $L_v = \bigcup_\varphi L_\varphi$, where the union is taken over all faces $\varphi$ of $\mathcal{A}^{||}(P^*)$ associated with the leaves of $\Pi_v$. Similarly, we define $P_v \subseteq P$ to be the set of points (taken without repetition) associated with the edges in $\Pi_v$. Set $m_v = |P_v|$ and $n_v = |L_v|$. At any level of $\mathcal{T}$,

$$\sum_v m_v = O(m^2) \qquad \text{and} \qquad \sum_v n_v = O(n). \tag{6.1}$$

By construction, any point in $P \setminus P_v$ lies either above or below all pseudo-lines in $L_v$. We therefore add two points, one at $y = +\infty$ and another at $y = -\infty$, to each $P_v$ and compute $\mathcal{F}(L_v, P_v)$ at each node $v$ of $\mathcal{T}$, in a bottom-up manner. Let $\kappa_v$ be the complexity of $\mathcal{F}(L_v, P_v)$. For each leaf $w \in \mathcal{T}$, we compute the lower and upper envelopes of $L_w$ in $O(n_w \log n_w)$ time. (Clearly, any of our marked faces must lie above the upper envelope or below the lower envelope.) For each internal node $v \in \mathcal{T}$, with children $w$ and $z$, we compute $\mathcal{F}(L_v, P_v)$ from $\mathcal{F}(L_w, P_w)$ and $\mathcal{F}(L_z, P_z)$ in $O((\kappa_v + \kappa_w + \kappa_z + m_v + n_v)\log n)$ time, using the "red-blue-merge" algorithm proposed by Edelsbrunner, Guibas, and Sharir [21] (and adapted by Guibas, Sharir, and Sifrony [29] for curves). Let $H$ be the height of $\mathcal{T}$, and let $h(v)$ denote the height of a node $v$ in $\mathcal{T}$ (i.e., the length of the longest path from $v$ to a leaf in its subtree). The time spent by the red-blue-merge algorithm over all nodes of $\mathcal{T}$ is

$$\sum_{v\in\mathcal{T}} O((\kappa_v + m_v + n_v)\log n) = \sum_{i=1}^{H} \sum_{v\in\mathcal{T},\, h(v)=i} O((\kappa_v + m_v + n_v)\log n). \tag{6.2}$$

The combination lemma of [21] implies that, for a node $v$ with children $w, z$, one has

$$\kappa_v \le \kappa_w + \kappa_z + 4m_v + 6n_v.$$

Therefore, if $\mathcal{T}_v$ (resp., $\Lambda_v$) is the set of nodes (resp., leaves) in the subtree rooted at

$v$, then

$$\kappa_v \leq \sum_{u \in \Lambda_v} \kappa_u + \sum_{z \in \mathcal{T}_v} O(m_z + n_z). \tag{6.3}$$

Since we compute only the upper and lower envelopes of $L_u$ at each leaf $u$ of $\mathcal{T}$, $\kappa_u = O(n_u)$. Using (6.1), (6.3), and the bound on $\kappa_u$, we obtain

$$\begin{aligned}\sum_{v \in \mathcal{T},\, h(v)=i} \kappa_v + m_v + n_v &= \sum_{v \in \mathcal{T},\, h(v)=i} O\left(\sum_{u \in \Lambda_v} n_u + \sum_{z \in \mathcal{T}_v} (m_z + n_z)\right) \\ &= \sum_{j \leq i} \sum_{\substack{z \in \mathcal{T} \\ h(z)=j}} O(m_z + n_z) \\ &= O((m^2 + n)i).\end{aligned}$$

Hence, (6.2) can now be rewritten as

$$\sum_{i=1}^{H} O((m^2+n)i \log n) = O((m^2+n)\log^3 n).$$

Together with the time spent in computing $\mathcal{A}(P^*)$, the overall running time of the algorithm is $O(m^{2+\varepsilon} + n^{1+\varepsilon})$. As in section 5, using the batching technique, the running time can be reduced to $O(mn^{1/2+\varepsilon} + n^{1+\varepsilon})$.

Next, we use $(1/r)$-cuttings to obtain an algorithm for computing $\mathcal{F}(L, P)$, which is efficient for all values of $m$ and $n$, as in [6], and in the general spirit of the preceding section. That is, we choose a parameter $r < n$ and compute a $(1/r)$-cutting $\Xi$ of $L$, as described in section 5. For each cell $\tau \in \Xi$, we compute $\mathcal{F}(L_\tau, P_\tau)$ in $O((m_\tau\sqrt{n_\tau} + n_\tau)n_\tau^{\varepsilon})$ time, using the algorithm just described. In addition, we also compute, in $O(n_\tau \log^2 n_\tau)$ time [29], the unbounded face of the arcs obtained by clipping $L_\tau$ within $\tau$. After having computed this for all cells of $\Xi$, we stitch the faces together to obtain $\mathcal{F}(L, P)$ in a straightforward manner; see [6] for details. The time spent in computing $\Xi$ and computing $\mathcal{F}(L_\tau, P)$ for each cell $\tau \in \Xi$ is

$$\sum_{\tau} O\left(m_\tau (n/r)^{1/2+\varepsilon} + (n/r)^{1+\varepsilon}\right) + O(nr).$$

Again, choosing an appropriate value of $r$ and following the same analysis as in section 5, we obtain the following result.

THEOREM 6.1. *Let $L$ be a set of $n$ pseudo-lines that are extensions of circular or polynomial arcs of bounded degree that traverse a fixed vertical strip $W$ in the plane, and let $P$ be a set of $m$ points in $W$, none lying on any pseudo-line. One can compute $\mathcal{F}(L, P)$ in time $O(m^{2/3-\varepsilon}n^{2/3+2\varepsilon} + m^{1+\varepsilon} + n^{1+\varepsilon})$. If there are $\chi$ crossing pairs of curves in $L$, $\mathcal{F}(L, P)$ can be computed in time $O(m^{2/3-\varepsilon}\chi^{1/3+\varepsilon} + m^{1+\varepsilon} + n^{1+\varepsilon})$ for any $\varepsilon > 0$.*

**7. Cutting lenses.** Among our main motivations for studying arrangements of pseudo-lines were the problems of computing incidences between points and circles and computing marked faces in an arrangement of circles. The recent analysis of Marcus and Tardos [33], following those of Aronov and Sharir [11] and Agarwal et al. [9], shows that a collection of $n$ circles can be cut into $O(n^{3/2}\log n)$ pseudo-segment arcs, meaning that any pair of arcs intersect at most once. One can then apply known bounds for incidences between points and pseudo-segments to obtain the bound $O(m^{2/3}n^{2/3} + n^{3/2}\log n)$ on the number of incidences between $m$ points and $n$ circles. (As already noted, this bound can then be further refined for small values of $m$; see [9, 11] for details.) Our goal is to make this combinatorial analysis constructive, so as to obtain a comparably efficient algorithm for detecting, counting, or reporting these incidences. The first task that we face is to find, in time $O(n^{3/2+\varepsilon})$, a set of $O(n^{3/2+\varepsilon})$ points that cut the given circles into pseudo-segments. If two circles $\gamma, \gamma' \in C$ intersect, then the boundary of each of the three bounded faces of $\mathcal{A}(\{\gamma, \gamma'\})$ is called a *lens*. Our goal is thus to cut the circles in $C$ so that all lenses will be cut, i.e., a cut is made on at least one of the two edges of every lens.

The algorithm proceeds in two stages. In the first stage, we use standard range-searching techniques [5] to decompose the intersection graph of the circles in $C$ into a union of complete bipartite subgraphs $\{A_i \times B_i\}_i$ such that (see also [11])[2]

$$\sum_i (|A_i| + |B_i|)^{3/2} = O(n^{3/2+\varepsilon}). \tag{7.1}$$

In the second stage, we cut circles in each bipartite subgraph independently. Let $A$ be a set of "red" circles and $B$ a set of "blue" circles so that every red circle intersects every blue circle; set $m = |A| + |B|$. We cut circles in $A$ and $B$ into circular arcs such that all *bichromatic* lenses, i.e., lenses formed by a red circle and a blue circle, are cut. We describe a recursive algorithm, which uses cutting-based decompositions of $\mathbb{R}^2$ into pseudo-trapezoids, for making these cuts. We first cut each circle in $A \cup B$ at its leftmost and rightmost points, yielding $2m$ $x$-monotone semicircles. At each step, we have a pseudo-trapezoid $\tau$ and two sets $\Gamma$, $\Gamma'$ of $x$-monotone circular arcs clipped to within $\tau$. The arcs in $\Gamma$ and $\Gamma'$ lie on the circles in $A$ and $B$, respectively. Initially, $\Gamma$ (resp., $\Gamma'$) is the set of upper and lower semicircles in $A$ (resp., $B$), and $\tau$ is the entire plane.

LEMMA 7.1. *If the endpoints of all arcs in $\Gamma$ and $\Gamma'$ lie on $\partial\tau$, then we can determine, in $O((|\Gamma| + |\Gamma'|)\log^3 m)$ time, whether $\Gamma$ and $\Gamma'$ induce at least one bichromatic lens that lies entirely in the interior of $\tau$.*

*Proof.* We use a two-level divide-and-conquer algorithm to determine whether a lens of the desired form exists. We choose a point, say, the topmost point $\xi$, on $\partial\tau$, and cut $\partial\tau$ at $\xi$. This induces a linear ordering on the endpoints of the arcs in $\Gamma \cup \Gamma'$, obtained by sorting the endpoints along $\partial\tau \setminus \{\xi\}$ in counterclockwise direction. For each arc $\gamma \in \Gamma \cup \Gamma'$, we refer to its first endpoint (in this ordering) as the *birth* point and to the second point as the *death* point of $\gamma$. See Figure 7 (i). We denote these points by $\beta_\gamma$ and $\delta_\gamma$, respectively.

Let $\lambda$ be a lens formed between an arc $\gamma \in \Gamma$ and an arc $\gamma' \in \Gamma'$, and lying fully in the interior of $\tau$. Then the four points $\beta_\gamma$, $\delta_\gamma$, $\beta_{\gamma'}$, $\delta_{\gamma'}$ must appear along $\partial\tau \setminus \{\xi\}$ either in the order $\beta_\gamma$, $\beta_{\gamma'}$, $\delta_{\gamma'}$, $\delta_\gamma$, or in the order $\beta_{\gamma'}$, $\beta_\gamma$, $\delta_\gamma$, $\delta_{\gamma'}$. We describe an algorithm that tests for the existence of a lens of the first kind; lenses of the second kind are handled in a fully symmetric manner.

We construct a 2-level range-tree data structure. The first-level structure $\mathcal{T}$ is a minimum-height binary tree that stores at its leaves the arcs of $\Gamma$ in increasing counterclockwise order of their birth points along $\partial\tau \setminus \{\xi\}$. For each node $v$ of $\mathcal{T}$,

[2]The same bound $O(n^{3/2+\varepsilon})$ also holds for the smaller sum $\sum_i (|A_i| + |B_i|)$. See remark (iv) in the concluding section for the implications of this fact for potential improvements of the algorithm.

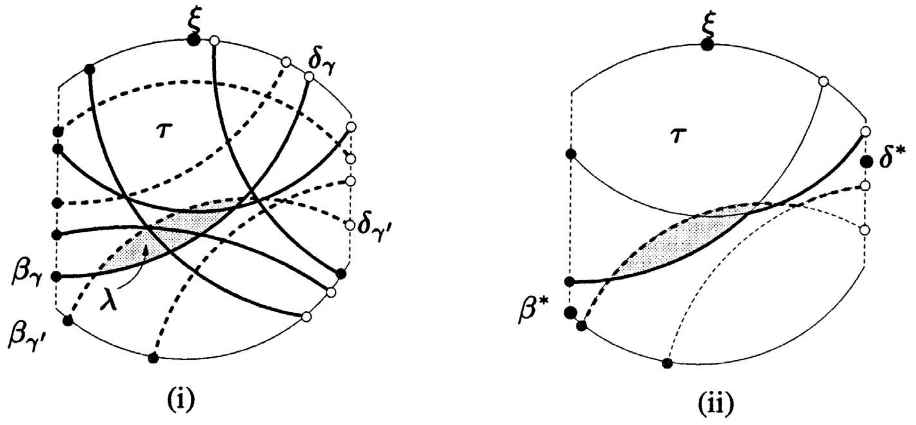


FIG. 7. (i) *All endpoints of* $\Gamma$ *(solid arcs) and* $\Gamma'$ *(dashed arcs) lie on* $\partial\tau$*; endpoints drawn as filled (resp., hollow) circles are birth (resp., death) points.* (ii) *A recursive subproblem: The arcs of* $\Gamma_{\mathcal{T}^*}$ *are drawn as solid and those of* $\Gamma'_{\mathcal{T}^*}$ *are drawn as dashed. The boundaries of the faces of* $\mathcal{A}(\Gamma_{\mathcal{T}^*})$ *and of* $\mathcal{A}(\Gamma'_{\mathcal{T}^*})$ *that contain* $\beta^*$ *and* $\delta^*$ *are drawn as bold. Since their overlay separates* $\beta^*$ *and* $\delta^*$*, this subproblem yields a bichromatic lens (two, as a matter of fact) fully contained in* $\tau$.

we construct a secondary tree $\mathcal{T}'_v$ on the set of all arcs that are stored at the leaves of the subtree rooted at $v$, and store them at the leaves of $\mathcal{T}'_v$, sorted in increasing counterclockwise order of their death points along $\partial\tau \setminus \{\xi\}$. We query with each arc $\gamma' \in \Gamma'$ in this data structure, as follows. We search in the primary tree for all arcs of $\gamma$ whose birth points precede the birth point of $\gamma'$ and obtain them as a collection of $O(\log m)$ disjoint subtrees. For each such subtree rooted at some node $v$, we go to the secondary structure $\mathcal{T}'_v$ and search for all arcs of $\Gamma$ that are stored there, whose death points succeed the death point of $\gamma'$, again obtaining them as a collection of $O(\log m)$ disjoint subtrees. Thus the query output consists of $O(\log^2 m)$ pairwise disjoint subtrees of secondary trees. We repeat this procedure for each arc $\gamma' \in \Gamma'$. This produces $O(m \log m)$ subproblems, where each subproblem is associated with a subtree $\mathcal{T}^*$ of some secondary tree $\mathcal{T}'_v$, and involves the set $\Gamma_{\mathcal{T}^*}$ of the arcs of $\Gamma$ stored at $\mathcal{T}^*$, and the set $\Gamma'_{\mathcal{T}^*}$ of those arcs of $\Gamma'$ for which $\mathcal{T}^*$ is one of their query output subtrees. The overall size $\sum_{\mathcal{T}^*}(|\Gamma'_{\mathcal{T}^*}| + |\Gamma'_{\mathcal{T}^*}|)$ of all subproblems is $O(m \log^2 m)$. Clearly, for any lens of the kind under consideration, formed by some $\gamma \in \Gamma$, $\gamma' \in \Gamma'$, there exists a (unique) subproblem involving both $\gamma$ and $\gamma'$.

Consider the subproblem associated with $\mathcal{T}^*$. By construction, there exist points $\beta^*, \delta^* \in \partial\tau$, such that all birth points of arcs in $\Gamma_{\mathcal{T}^*}$ lie between $\xi$ and $\beta^*$ (in counterclockwise direction), all birth and death points of arcs in $\Gamma'_{\mathcal{T}^*}$ lie between $\beta^*$ and $\delta^*$, and all death points of arcs in $\Gamma_{\mathcal{T}^*}$ lie between $\delta^*$ and $\xi$. It is easy to see that $\beta^*$ and $\delta^*$ lie below the lower envelope of $\Gamma_{\mathcal{T}^*}$ and above the upper envelope of $\Gamma'_{\mathcal{T}^*}$. Hence the subproblem of $\mathcal{T}^*$ induces a lens of the desired kind if and only if $\beta^*$ and $\delta^*$ lie in different connected components of the sandwich region formed by the lower envelope of $\Gamma_{\mathcal{T}^*}$ and the upper envelope of $\Gamma'_{\mathcal{T}^*}$, clipped to within $\tau$. See Figure 7 (ii). We can compute this sandwich region in $O((|\Gamma_{\mathcal{T}^*}| + |\Gamma'_{\mathcal{T}^*}|) \log m)$ time, using the fact that all arcs are $x$-monotone and circular [38]. Summing this bound over all subtrees $\mathcal{T}^*$, the lemma follows. □

If the endpoints of all arcs in $\Gamma$ and $\Gamma'$ lie on $\partial\tau$ and $\Gamma$ and $\Gamma'$ do not induce a lens that lies completely inside $\tau$, we stop. Otherwise (i.e., either an endpoint lies inside $\tau$, or there is a bichromatic lens lying fully inside $\tau$), we choose a sufficiently large constant $r$ and compute, within $\tau$, a $(1/r)$-cutting $\Xi$ of $\Gamma \cup \Gamma'$, of size $cr^2$, for a constant $c > 0$, as described in section 5. For every arc $\gamma \in \Gamma \cup \Gamma'$ and for every cell $\Delta \in \Xi$ that is crossed by $\gamma$, we cut $\gamma$ at its intersection points with $\partial\Delta$. The total number of cuts made is $O(mr)$. After this step, all lenses that lie in more than one cell of $\Xi$ have been cut, so we recursively solve the problem within each cell $\Delta$ of $\Xi$, with the sets $\Gamma_\Delta$ and $\Gamma'_\Delta$, where $\Gamma_\Delta$ is the set of the connected components of the intersections $\gamma \cap \Delta$, for $\gamma \in \Gamma$, and where $\Gamma'_\Delta$ is similarly defined.

It is clear that the algorithm cuts all bichromatic lenses inside $\tau$. (Initially, $\tau$ is the whole plane.) In order to analyze its performance, we need the following result, proven in [9, Lemma 3.2] and stated here in a slightly different manner.

Lemma 7.2 (see [9]). *Let $C = \Gamma \cup \Gamma'$ be a family of $n$ pseudo-circles, where each pseudo-circle in $\Gamma$ intersects every pseudo-circle in $\Gamma'$ twice. Then the maximum size of a family of pairwise-nonoverlapping bichromatic lenses in $\mathcal{A}(C)$ with pairwise disjoint interiors is $O(n)$.*

Returning to the subproblem inside the trapezoid $\tau$, let $m = |\Gamma| + |\Gamma'|$, let $k$ be the number of endpoints of arcs in $\Gamma \cup \Gamma'$ that lie in the interior of $\tau$ plus the maximum size of a family of pairwise-nonoverlapping bichromatic lenses in $\mathcal{A}(C)$ with pairwise disjoint interiors that are fully contained in the interior of $\tau$,[3] and let $N(m, k)$ denote the maximum number of cuts that the above algorithm performs, for sets $\Gamma, \Gamma'$ with parameters $m$ and $k$. Since the algorithm does not make any cuts if neither an endpoint nor a bichromatic lens lies inside $\tau$, we have $N(m, 0) = 0$. For $k > 0$, $N(m, k)$ satisfies the recurrence

$$N(m,k) \le \sum_{\Delta \in \Xi} N(m/r, k_\Delta) + amr,$$

where $a$ is a constant and $k_\Delta$ is the number of endpoints of $\Gamma \cup \Gamma'$ that lie in the interior of a cell $\Delta$ plus the maximum size of a family of pairwise-nonoverlapping bichromatic lenses in $\mathcal{A}(C)$ with pairwise disjoint interiors that are fully contained in the interior of $\Delta$. By definition, we have $\sum_\Delta k_\Delta \le k$. We claim that

$$N(m,k) \le Am^{1+\varepsilon}\sqrt{k}$$

for some constant $A$ that depends on $\varepsilon$. Indeed, the bound trivially holds if either $m$ or $k$ is 0. For larger values of $m$ and $k$, we have, by induction hypothesis (here $c$ is the constant defined above so that $cr^2$ is an upper bound on the size of $\Xi$),

$$\begin{aligned} N(m,k) &\le \sum_{\Delta \in \Xi} A\left(\frac{m}{r}\right)^{1+\varepsilon}\sqrt{k_\Delta} + amr \\ &\le Am^{1+\varepsilon}\frac{\sqrt{\sum_\Delta k_\Delta}\sqrt{|\Xi|}}{r^{1+\varepsilon}} + amr \\ &\le Am^{1+\varepsilon}\sqrt{k}\left(\frac{\sqrt{c}}{r^\varepsilon} + \frac{ar}{Am^\varepsilon\sqrt{k}}\right) \\ &\le Am^{1+\varepsilon}\sqrt{k}, \end{aligned}$$

provided that $r > (2\sqrt{c})^{1/\varepsilon}$ and $A \ge 2ar$. By Lemma 7.2, we have $k = O(m)$. Hence, the algorithm makes $O(m^{3/2+\varepsilon})$ cuts.

[3]Note that $k$ is not known to the algorithm and is used only for the purpose of the analysis of its performance.

Finally, let $T(m,k)$ denote the maximum running time of the above algorithm for sets $\Gamma, \Gamma'$ that have parameters $m$ and $k$. Then, by Lemma 7.1, $T(m,0) = O(m \log^3 m)$. As in the analysis of $N(m,k)$, we obtain the following recurrence for $T(m,k)$:

$$T(m,k) = \sum_{\Delta \in \Xi} T(m/r, k_\Delta) + O(m \log^3 m).$$

The same analysis as for $N(m,k)$ implies that

$$T(m,k) = O(m^{1+\varepsilon}\sqrt{k}).$$

Substituting $k = O(m)$, the maximum running time of the algorithm is $O(m^{3/2+\varepsilon})$.

Repeating this procedure to all bipartite graphs $A_i \times B_i$ and adding up the resulting complexity bounds, using (7.1), we obtain the following theorem.

THEOREM 7.3. *A collection of $n$ circles in $\mathbb{R}^2$ can be cut into $O(n^{3/2+\varepsilon})$ pseudo-segments, in time $O(n^{3/2+\varepsilon})$, for any $\varepsilon > 0$.*

**8. Circle arrangements.** Let $\mathcal{C}$ be a set of $n$ circles and $P$ a set of $m$ points in $\mathbb{R}^2$, and let $\chi$ be the number of intersection points between the circles of $\mathcal{C}$. We combine the algorithm described in section 7 with those in sections 5 and 6 to obtain efficient algorithms for computing $\mathcal{I}(\mathcal{C}, P)$ and $\mathcal{F}(\mathcal{C}, P)$. This is done as follows.

Using Theorem 7.3, we cut $\mathcal{C}$ into a set $L$ of $N = O(n^{3/2+\varepsilon})$ pseudo-segments for any $\varepsilon > 0$. Each pair of arcs in $L$ intersect at most once, but the endpoints of these arcs do not lie on the boundary of a common vertical strip. To adapt the algorithms of sections 5 and 6, we construct a segment tree $\mathcal{T}$ on the $x$-projections of the arcs in $L$. Each node $v$ of $\mathcal{T}$ is associated with a vertical strip $W_v$. Let $L_v$ be the subset of arcs in $L$, clipped within $W_v$, that are stored at $v$ (these arcs cross $W_v$ from left to right, but have an endpoint inside the parent strip), and let $\bar{L}_v$ be the set of arcs that are stored at the descendants of $v$ (including $v$ itself), again clipped within $W_v$ (each of these arcs that is stored at a proper descendant has an endpoint in the interior of $W_v$). Let $P_v = W_v \cap P$. Set $N_v = |L_v|, \bar{N}_v = |\bar{L}_v|$, and $m_v = |P_v|$. Let $\chi_v$ (resp., $\bar{\chi}_v$) be the number of intersections between the arcs of $L_v$ (resp., of $\bar{L}_v$). We construct $\mathcal{I}(L,P)$ and $\mathcal{F}(L,P)$ by visiting the nodes of $\mathcal{T}$ in a bottom-up manner and computing $\mathcal{I}(\bar{L}_v, P_v)$ and $\mathcal{F}(\bar{L}_v, P_v)$ at each node $v \in \mathcal{T}$. If $\xi$ is the root of $\mathcal{T}$, then $\mathcal{I}(L,P) = \mathcal{I}(\bar{L}_\xi, P_\xi)$ and $\mathcal{F}(L,P) = \mathcal{F}(\bar{L}_\xi, P_\xi)$.

*Computing incidences.* Let $w$ and $z$ be the two children of a node $v$ in $\mathcal{T}$. Then

$$\mathcal{I}(\bar{L}_v, P_v) = \mathcal{I}(L_v \cup \bar{L}_w \cup \bar{L}_z, P_v) = \mathcal{I}(L_v, P_v) \cup \mathcal{I}(\bar{L}_w, P_w) \cup \mathcal{I}(\bar{L}_z, P_z).$$

The last equality holds because $P_w \cap W_z = P_z \cap W_w = \emptyset$. The sets $\mathcal{I}(\bar{L}_w, P_w)$ and $\mathcal{I}(\bar{L}_z, P_z)$ are computed recursively, so we need only to compute $\mathcal{I}(L_v, P_v)$ at $v$. Since the endpoints of all arcs in $L_v$ lie on the boundary of $W_v$, we can compute $\mathcal{I}(L_v, P_v)$ in time $O(m_v^{2/3-\varepsilon}\chi_v^{1/3+\varepsilon} + m_v^{1+\varepsilon} + N_v^{1+\varepsilon})$, for any $\varepsilon > 0$, using Theorem 5.2. Using the fact that $\sum_v m_v = O(m \log m)$, $\sum_v N_v = O(N \log N)$, and $\sum_v \chi_v \le \chi$, the overall running time, summed over all nodes of $\mathcal{T}$, is

$$O(m^{2/3-\varepsilon}\chi^{1/3+\varepsilon} + m^{1+\varepsilon} + n^{3/2+\varepsilon})$$

for any $\varepsilon > 0$, which has to be chosen larger than the preceding $\varepsilon$ but can still be arbitrarily small.

This bound is nearly worst-case optimal for $m \geq n^{5/4}$. For smaller values of $m$, we can improve the algorithm, following the decomposition technique of [9, 11]. Specifically, we map each circle $C \in \mathcal{C}$ to a point $C^* \in \mathbb{R}^3$ and each point $p \in \mathbb{R}^2$ to a plane $p^*$ in $\mathbb{R}^3$, so that $p$ lies inside (resp., on, outside) $C$ if and only if $C^*$ lies above (resp., on, below) $p^*$. Let $\mathcal{C}^* = \{C^* \mid C \in \mathcal{C}\}$ and $P^* = \{p^* \mid p \in P\}$. We choose the parameter $r = \min\left\{\lceil n^{5/11}/m^{4/11}\rceil, m\right\}$ and compute, in $O(mr^2)$ time, a hierarchical $(1/r)$-cutting $\Xi$ of $P^*$, as mentioned in section 5. For each cell $\Delta \in \Xi$, let $\mathcal{C}^*_\Delta = \mathcal{C}^* \cap \Delta$, let $P^*_\Delta \subseteq P^*$ be the set of planes that cross $\Delta$, and let $\bar{P}^*_\Delta$ be the set of planes that contain $\Delta$. Chazelle's technique computes $\mathcal{C}^*_\Delta$ and $P^*_\Delta$ explicitly, for all cells $\Delta$, in time $O(n \log r + mr^2)$ [17]. Moreover, for any cell $\Delta$, $\bar{P}^*_\Delta$ can be computed in time $O(\log m + |\bar{P}^*_\Delta|)$ by traversing the ancestors of $\Delta$ (i.e., the cells in the intermediate cuttings that contain $\Delta$). Let $C_\Delta$, $P_\Delta$, $\bar{P}_\Delta$ denote the corresponding primal sets of circles and points. We compute $\mathcal{I}(\mathcal{C}_\Delta, P_\Delta)$ in time

$$O(m_\Delta^{2/3-\varepsilon}\chi_\Delta^{1/3+\varepsilon} + m_\Delta^{1+\varepsilon} + n_\Delta^{3/2+\varepsilon})$$

using the algorithm just outlined, where $m_\Delta = |P_\Delta|$, $n_\Delta = |\mathcal{C}_\Delta|$, and $\chi_\Delta$ is the number of intersection points between the circles of $\mathcal{C}_\Delta$. We also report all pairs in $\bar{P}_\Delta \times \mathcal{C}_\Delta$. Since $m_\Delta \leq m/r$ for each $\Delta$, $\sum_\Delta n_\Delta = n$, and $\sum_\Delta \chi_\Delta \leq \chi$, we obtain the following result by a straightforward calculation (see also [9, 11]).

THEOREM 8.1. *Let $\mathcal{C}$ be a set of $n$ circles in $\mathbb{R}^2$ and $P$ a set of $m$ points in $\mathbb{R}^2$, and let $\chi$ be the number of intersection points between the circles of $\mathcal{C}$. Then the incidences between $\mathcal{C}$ and $P$ can be detected, counted, or reported, in time $O(m^{2/3-\varepsilon}\chi^{1/3+\varepsilon} + m^{6/11+3\varepsilon}\chi^{4/11+2\varepsilon}n^{1/11-5\varepsilon} + m^{1+\varepsilon} + n^{1+\varepsilon})$, for any $\varepsilon > 0$.*

*Computing marked faces.* Here the points of $P$ do not lie on any circle of $\mathcal{C}$ and are used to mark faces of $\mathcal{A}(\mathcal{C})$ that we wish to construct. To do so, we follow the same approach as for computing $\mathcal{I}(L, P)$, but we need to be a little more careful. Consider the processing of a node $v$ of $T$. For a subset $X \subseteq \bar{L}_v$ and a set $A$ of points, let $\bar{\mathcal{F}}(X, A)$ denote $\mathcal{F}(X, A)$ together with the unbounded face of $\mathcal{A}(X)$. (Since the arcs in $\bar{L}_v$ are clipped to within $W_v$, there is a single unbounded face.) Let $w$ and $z$ be the children of $v$. Then we have $\bar{\mathcal{F}}(\bar{L}_w, P_v) = \bar{\mathcal{F}}(\bar{L}_w, P_w)$ and $\bar{\mathcal{F}}(\bar{L}_z, P_v) = \bar{\mathcal{F}}(\bar{L}_z, P_z)$. Moreover, $\bar{\mathcal{F}}(\bar{L}_w \cup \bar{L}_z, P_v)$ can be obtained from $\bar{\mathcal{F}}(\bar{L}_w, P_w)$ and $\bar{\mathcal{F}}(\bar{L}_z, P_z)$ by simply stitching together the common endpoints of the clipped arcs that they share at the common boundary of $W_w$ and $W_z$, updating the representation and the global structure of the faces as this stitching takes place.

We recursively compute $\bar{\mathcal{F}}(\bar{L}_w, P_w)$ and $\bar{\mathcal{F}}(\bar{L}_z, P_z)$ and merge them into $\bar{\mathcal{F}}(\bar{L}_w \cup \bar{L}_z, P_v)$ by stitching the subarcs as just described. We then compute $\bar{\mathcal{F}}(L_v, P_v)$ using Theorem 6.1, and merge the two collections of faces, $\bar{\mathcal{F}}(L_v, P_v)$ and $\bar{\mathcal{F}}(\bar{L}_w \cup \bar{L}_z, P_v)$, using the red-blue-merge algorithm mentioned in section 6, to obtain $\bar{\mathcal{F}}(\bar{L}_v, P_v)$. The total time spent at $v$ is $O(m_v^{2/3-\varepsilon}\chi_v^{1/3+\varepsilon} + m_v^{1+\varepsilon} + N_v^{1+\varepsilon})$. Summing up this cost over all nodes of $\mathcal{T}$, the total time spent in computing $\mathcal{F}(L, P)$ is

$$O(m^{2/3-\varepsilon}\chi^{1/3+\varepsilon} + m^{1+\varepsilon} + n^{3/2+\varepsilon}).$$

Note that at the root $\xi$ we also obtain the unbounded face of $\mathcal{A}(C)$, regardless of whether it is one of the marked faces.

For small values of $m$, as in the case of incidences, we compute the sets $\mathcal{C}^*$ and $P^*$, choose the parameter $r = \lceil n^{5/11}/m^{4/11}\rceil$, construct a hierarchical $(1/r)$-cutting $\Xi$ of $\mathcal{A}(P^*)$ in $\mathbb{R}^3$, and obtain the subsets $P_\Delta$, $\mathcal{C}_\Delta$, for each cell $\Delta \in \Xi$, as defined above (because of the generic locations of the points in $P$, the sets $\bar{P}_\Delta$ are not needed in the case of marked faces). For each $\Delta \in \Xi$, any point in $P \setminus P_\Delta$ lies either in the common interior of all the circles in $\mathcal{C}_\Delta$ or in their common exterior. For each $\Delta$, we compute $\mathcal{F}(\mathcal{C}_\Delta, P_\Delta)$ in time

$$O(m_\Delta^{2/3-\varepsilon}\chi_\Delta^{1/3+\varepsilon} + m_\Delta^{1+\varepsilon} + n_\Delta^{3/2+\varepsilon}).$$

We next compute the common interior and the common exterior of $\mathcal{C}_\Delta$. These additional faces have a combined complexity of only $O(n_\Delta)$ [31], and they can be constructed in randomized expected time $O(n_\Delta \log n_\Delta)$ [36] or in deterministic time $O(n_\Delta \log^2 n_\Delta)$ [31]. Adding these faces to $\mathcal{F}(\mathcal{C}_\Delta, P_\Delta)$, we thus obtain $\bar{\mathcal{F}}(\mathcal{C}_\Delta, P_\Delta) \supseteq \mathcal{F}(\mathcal{C}_\Delta, P)$. Plugging in the value of $r$ and using the fact that $m_\Delta \le m/r$ and $n_\Delta \le n/r^3$, we obtain that the total time spent so far is

$$O(m^{2/3-\varepsilon}n^{2/3+2\varepsilon} + m^{6/11+3\varepsilon}n^{9/11-\varepsilon} + m^{1+\varepsilon} + n^{1+\varepsilon}).$$

Recall (cf. section 5) that a hierarchical cutting consists of a sequence of cuttings $\Xi_0, \Xi_1, \ldots, \Xi_\nu = \Xi$, where $\Xi_i$ is a $(1/r_0^i)$-cutting of $P^*$ for some constant $r_0$ and $\nu = \lceil \log_{r_0} r \rceil$. We extend the notation $P_\tau$ and $\mathcal{C}_\tau$ to the cells $\tau$ of the intermediate cuttings $\Xi_i$. If $\tau \in \Xi_i$, then $m_\tau = |P_\tau| \le m/r_0^i$, $n_\tau = |\mathcal{C}_\tau| \le n/r_0^{3i}$, and $\sum_{\tau\in\Xi_i} n_\tau \le n$. For each $\tau \in \Xi_i$, the next cutting $\Xi_{i+1}$ contains a $(1/r_0)$-cutting $\Xi^{(\tau)}$ of $P_\tau^*$ (with respect to $\tau$) of size at most $O(r_0^3)$. By proceeding in decreasing order of $i$, we compute $\mathcal{F}(\mathcal{C}_\tau, P)$, for each $\tau \in \Xi_i$, as follows: If $i = \nu$, we have already computed the collections of faces $\mathcal{F}(\mathcal{C}_\tau, P)$. If $i < \nu$, let $\tau_1, \ldots, \tau_s$ be the simplices in $\Xi^{(\tau)}$. We compute $\mathcal{F}(\mathcal{C}_\tau, P)$ from $\mathcal{F}(\mathcal{C}_{\tau_1}, P_{\tau_1}), \ldots, \mathcal{F}(\mathcal{C}_{\tau_s}, P_{\tau_s})$, which we have already computed, by using the red-blue-merge algorithm repeatedly. Let $\kappa_\tau$ be the complexity of $\mathcal{F}(\mathcal{C}_\tau, P)$. Since all points in $P \setminus P_\tau$ lie either in the common exterior or in the common interior of all circles of $C_\tau$, the total time spent in computing $\mathcal{F}(\mathcal{C}_\tau, P)$ is

$$O\left(\left(\kappa_\tau + \sum_{i=1}^{s} \kappa_{\tau_i} + m_\tau + n_\tau\right)\log n\right),$$

where the constant of proportionality depends on $r_0$.[4] Since $\Xi_0$ consists of a single cell, namely, the entire $\mathbb{R}^3$, we have $\mathcal{F}(\mathcal{C}, P)$ at our disposal after we have processed the unique cell of $\Xi_0$.

Summing the cost over all cells of the intermediate cuttings, the total running time is

$$\begin{aligned}
&\sum_{i=0}^{\nu}\sum_{\tau\in\Xi_i} O\left(\left(\kappa_\tau + \sum_{\tau'\in\Xi^{(\tau)}} \kappa_{\tau'} + m_\tau + n_\tau\right)\log n\right)\\
&= \sum_{i=0}^{\nu}\sum_{\tau\in\Xi_i} O(\kappa_\tau \log n) + \sum_{i=0}^{\nu} O((m r_0^{2i} + n)\log n)\\
&= \sum_{i=0}^{\nu}\sum_{\tau\in\Xi_i} O(\kappa_\tau \log n) + O((mr^2 + n\log r)\log n). \qquad (8.1)
\end{aligned}$$

[4]This is the main reason for using hierarchical cuttings here: Applying the red-blue-merge for a single-level cutting, with a nonconstant value of $r$, would result in a constant of proportionality that is a high power of $r$, and would thus be too expensive.

As mentioned in the introduction, the recent result of Agarwal, Aronov, and Sharir [2], enhanced by the more recent result of Marcus and Tardos [33], implies that

$$\kappa_\tau = O(m_\tau^{2/3}n_\tau^{2/3} + m_\tau^{6/11}n_\tau^{9/11}\log^{6/11}(m_\tau^3/n_\tau) + n_\tau \log n_\tau).$$

Substituting the individual bounds on $m_\tau$ and $n_\tau$, we obtain

$$\begin{aligned}\sum_{i=0}^{\nu}\sum_{\tau\in\Xi_i}\kappa_\tau &= \sum_{i=0}^{\nu}\sum_{\tau\in\Xi_i} O(m_\tau^{2/3}n_\tau^{2/3} + m_\tau^{6/11}n_\tau^{9/11}\log^{6/11}(m_\tau^3/n_\tau) + n_\tau\log n_\tau)\\ &= \sum_{i=0}^{\nu} O(m^{2/3}n^{2/3}r_0^{i/3} + m^{6/11}n^{9/11}\log^{6/11}(m) + n\log n)\\ &= O(m^{2/3}n^{2/3}r^{1/3} + m^{6/11}n^{9/11}\log^{6/11}(m) + n\log^2 n). \qquad (8.2)\end{aligned}$$

Plugging (8.2) and the value of $r$ into (8.1), the total running time is

$$O((m^{2/3}n^{2/3} + m^{6/11}n^{9/11}\log^{6/11}(m)\log n + n\log^2 n)\log n).$$

Adding the time spent in computing $\mathcal{F}(\mathcal{C}_\Delta, P)$ for all cells $\Delta$ in the final cutting $\Xi$, we obtain the following result.

THEOREM 8.2. *Given a set $\mathcal{C}$ of $n$ circles and a set $P$ of $m$ points in $\mathbb{R}^2$, we can compute $\mathcal{F}(\mathcal{C}, P)$ in time $O(m^{2/3-\varepsilon}n^{2/3+2\varepsilon} + m^{6/11+3\varepsilon}n^{9/11-\varepsilon} + n^{1+\varepsilon})$ for any $\varepsilon > 0$.*

As for the case of incidences, we can obtain a bound that is sensitive to the number of intersecting pairs of circles in $\Xi$. Omitting the details, we obtain the following result.

THEOREM 8.3. *Given a set $\mathcal{C}$ of $n$ circles with $\chi$ intersecting pairs and a set $P$ of $m$ points in $\mathbb{R}^2$, we can compute $\mathcal{F}(\mathcal{C}, P)$ in time*

$$O(m^{2/3-\varepsilon}\chi^{1/3+\varepsilon} + m^{6/11+3\varepsilon}\chi^{4/11+2\varepsilon}n^{1/11-5\varepsilon} + n^{1+\varepsilon})$$

*for any $\varepsilon > 0$.*

*Remark* 8.4. Theorems 8.1–8.3 do not extend to pseudo-circles for small values of $m$ because we map $\mathcal{C}$, the family of input circles, to a set $P^*$ of points in $\mathbb{R}^3$ by using the so-called lifting transform, which we do not know how to extend to pseudo-circles. See an expanded remark in the next section.

*Handling congruent circles.* If $\mathcal{C}$ is a set of congruent circles, then the algorithm for computing $\mathcal{F}(\mathcal{C}, P)$ can be improved and simplified. We partition each circle in $\mathcal{C}$ into two semicircles by splitting it at its leftmost and rightmost points. Let $U$ and $L$ denote the sets of resulting upper and lower semicircles, respectively. Each pair of arcs within $U$ (or within $L$) intersect in at most one point. We use the segment-tree based algorithm just described to compute $\mathcal{F}(L, P)$ and $\mathcal{F}(U, P)$, in time $O(m^{2/3-\varepsilon}\chi^{1/3+\varepsilon} + m^{1+\varepsilon} + n^{1+\varepsilon})$. We can then compute $\mathcal{F}(\mathcal{C}, P) = \mathcal{F}(U \cup L, P)$ from $\mathcal{F}(L, P)$ and $\mathcal{F}(U, P)$ in time $O(\kappa \log n)$ using, as above, the red-blue-merge algorithm of [21], where $\kappa$ is the total number of vertices in $\mathcal{F}(L, P)$, $\mathcal{F}(U, P)$, and $\mathcal{F}(\mathcal{C}, P)$. Hence we obtain the following result.

THEOREM 8.5. *Let $P$ be a set of $m$ points and $\mathcal{C}$ a set of $n$ congruent circles with $\chi$ intersecting pairs in $\mathbb{R}^2$. We can compute $\mathcal{F}(\mathcal{C}, P)$ in time $O(m^{2/3-\varepsilon}\chi^{1/3+\varepsilon} + m^{1+\varepsilon} + n^{1+\varepsilon})$ for any $\varepsilon > 0$.*

## 9. Conclusion.

In this paper we define a duality transform for pseudo-lines and present an efficient algorithm for constructing the dual arrangement for the case when the pseudo-lines are defined by circular arcs or graphs of polynomials of bounded degree, clipped to within some vertical strip. Our algorithm relies on a dynamic pseudo-halfplane-emptiness data structure, an extension of the halfspace-emptiness data structure of Matoušek [34], and is therefore interesting in its own right. Finally, we present various applications of our duality algorithm. Roughly speaking, our algorithm extends algorithms for many problems involving arrangements of lines to arrangements of pseudo-lines, which in turn leads to faster algorithms for problems involving arrangements of circles and of other arcs. We conclude by mentioning a few open problems.

(i) It is known [13, 23] that there does not exist a duality transform between points and pseudo-hyperplanes in higher dimensions that preserves the above-below relation. However it is not known whether the *lifting transform*, used in section 8, can be extended to pseudo-circles. That is, given a set $\mathcal{C}$ of pseduo-circles and a set $P$ of points in the plane, can we map each $C \in \mathcal{C}$ to a point $C^*$ in $\mathbb{R}^3$ and each point $p \in P$ to a pseudo-plane $p^*$ in $\mathbb{R}^3$ so that $p$ lies outside $C$ if and only if $C^*$ lies below $p^*$? Such a transform, and an efficient algorithm for computing it, will allow us to extend several known algorithms for circle arrangements (e.g., computing incidences and computing a family of marked faces) to pseudo-circle arrangements.

(ii) Currently our pseudo-halfplane-emptiness data structure, and thus our algorithm for constructing the dual arrangement, does not work efficiently for algebraic arcs defined by implicit equations, e.g., a family of $x$-monotone elliptic arcs whose endpoints lie on the boundary of a fixed vertical strip. We need to extend Lemma 4.2 to such arcs in order to answer pseudo-halfplane-emptiness queries efficiently.

(iii) Since our lens-cutting algorithm uses a circle-range-searching structure in its first step, it does not extend to pseudo-circles. It would be useful to develop an efficient algorithm that bypasses the range-searching step.

(iv) The current combinatorial bound on point-circle incidences is not believed to be optimal for smaller point sets. Should that bound be improved, the current method would still need a more efficient algorithm that cuts circles into fewer pseudo-segments. The current approach, based on complete bipartite decomposition of the intersection graph of circles, is "stuck" with near-$n^{3/2}$ running time.

**Acknowledgments.** The authors thank Boris Aronov and Sariel Har-Peled for useful discussions, Eli Goodman for pointing out several relevant references, and two referees for their helpful comments.

## REFERENCES

[1] P. K. Agarwal, *Partitioning arrangements of lines.* II. *Applications*, Discrete Comput. Geom., 5 (1990), pp. 533–573.

[2] P. K. Agarwal, B. Aronov, and M. Sharir, *On the complexity of many faces in arrangements of pseudo-segments and of circles*, in Discrete and Computational Geometry, The Goodman-Pollack Festschrift, Algorithms Combin. 25, B. Aronov, S. Basu, J. Pach, and M. Sharir, eds., Springer-Verlag, Berlin, 2003, pp. 1–24.

[3] P. K. Agarwal, B. Aronov, M. Sharir, and S. Suri, *Selecting distances in the plane*, Algorithmica, 9 (1993), pp. 495–514.

[4] P. K. Agarwal, A. Efrat, and M. Sharir, *Vertical decomposition of shallow levels in* 3*-dimensional arrangements and its applications*, SIAM J. Comput., 29 (1999), pp. 912–953.

[5] P. K. Agarwal and J. Erickson, *Geometric range searching and its relatives*, in Advances in Discrete and Computational Geometry, Contemp. Math. 223, B. Chazelle, J. E. Goodman, and R. Pollack, eds., AMS, Providence, RI, 1999, pp. 1–56.

[6] P. K. Agarwal, T. Hagerup, R. Ray, M. Sharir, M. Smid, and E. Welzl, *Translating a planar object to maximize point containment: Exact and approximation algorithms*, in Proceedings of the 10th Annual European Symposium on Algorithms, Lecture Notes in Comput. Sci. 2461, Springer-Verlag, London, 2002, pp. 42–53.

[7] P. K. Agarwal and J. Matoušek, *On range searching with semialgebraic sets*, Discrete Comput. Geom., 11 (1994), pp. 393–418.

[8] P. K. Agarwal and J. Matoušek, *Dynamic half-space range reporting and its applications*, Algorithmica, 13 (1995), pp. 325–345.

[9] P. K. Agarwal, E. Nevo, J. Pach, R. Pinchasi, M. Sharir, and S. Smorodinsky, *Lenses in arrangements of pseudo-circles and their applications*, J. ACM, 51 (2004), pp. 606–635.

[10] P. K. Agarwal and M. Sharir, *Arrangements and their applications*, in Handbook of Computational Geometry, J.-R. Sack and J. Urrutia, eds., North–Holland, Amsterdam, 2000, pp. 49–119.

[11] B. Aronov and M. Sharir, *Cutting circles into pseudo-segments and improved bounds for incidences*, Discrete Comput. Geom., 28 (2002), pp. 475–490.

[12] J. L. Bentley and T. A. Ottmann, *Algorithms for reporting and counting geometric intersections*, IEEE Trans. Comput., C-28 (1979), pp. 643–647.

[13] A. Björner, M. Las Vergnas, N. White, B. Sturmfels, and G. M. Ziegler, *Oriented Matroids*, Cambridge University Press, Cambridge, UK, 1993.

[14] S. A. Burr, B. Grünbaum, and N. J. A. Sloane, *The orchard problem*, Geom. Dedicat., 2 (1974), pp. 397–424.

[15] T. M. Chan, *On levels in arrangements of curves*, Discrete Comput. Geom., 29 (2003), pp. 275–293.

[16] T. M. Chan, *On levels in arrangements of curves,* II*: A simple inequality and its consequences*, in Proceedings of the 44th Annual IEEE Symposium on Foundations of Computer Science (Cambridge, MA, 2003), IEEE Computer Society Press, Los Alamitos, CA, 2003, pp. 544–550.

[17] B. Chazelle, *Cutting hyperplanes for divide-and-conquer*, Discrete Comput. Geom., 9 (1993), pp. 145–158.

[18] K. Clarkson, H. Edelsbrunner, L. J. Guibas, M. Sharir, and E. Welzl, *Combinatorial complexity bounds for arrangements of curves and spheres*, Discrete Comput. Geom., 5 (1990), pp. 99–160.

[19] M. de Berg, M. van Kreveld, M. Overmars, and O. Schwarzkopf, *Computational Geometry: Algorithms and Applications*, Springer-Verlag, Berlin, 1997.

[20] H. Edelsbrunner, *Algorithms in Combinatorial Geometry*, EATCS Monographs on Theoretical Computer Science 10, Springer-Verlag, Berlin, 1987.

[21] H. Edelsbrunner, L. J. Guibas, and M. Sharir, *The complexity and construction of many faces in arrangements of lines and of segments*, Discrete Comput. Geom., 5 (1990), pp. 161–196.

[22] J. E. Goodman, *Proof of a conjecture of Burr, Grünbaum and Sloane*, Discrete Math., 32 (1980), pp. 27–35.

[23] J. E. Goodman, *Pseudoline arrangements*, in Handbook of Discrete and Computational Geometry, J. E. Goodman and J. O'Rourke, eds., CRC, Boca Raton, FL, 1997, pp. 83–109.

[24] J. E. Goodman and R. Pollack, *On the combinatorial classification of nondegenerate configurations in the plane*, J. Combin. Theory Ser. A, 29 (1980), pp. 220–235.

[25] J. E. Goodman and R. Pollack, *A theorem of ordered duality*, Geom. Dedicata, 12 (1982), pp. 63–74.

[26] J. E. Goodman and R. Pollack, *Allowable sequences and order types in discrete and computational geometry*, in New Trends in Discrete and Computational Geometry, Algorithms Combin. 10, J. Pach, ed., Springer-Verlag, Berlin, 1993, pp. 103–134.

[27] B. Grünbaum, *Arrangements of hyperplanes*, Congr. Numer., 3 (1971), pp. 41–106.

[28] B. Grünbaum, *Arrangements and Spreads*, Regional Conference Series Math. 10, AMS, Providence, RI, 1972.

[29] L. J. Guibas, M. Sharir, and S. Sifrony, *On the general motion planning problem with two degrees of freedom*, Discrete Comput. Geom., 4 (1989), pp. 491–521.

[30] S. Har-Peled and M. Sharir, *On-line point location in planar arrangements and its applications*, Discrete Comput. Geom., 26 (2001), pp. 19–40.

[31] K. Kedem, R. Livne, J. Pach, and M. Sharir, *On the union of Jordan regions and collision-free translational motion amidst polygonal obstacles*, Discrete Comput. Geom., 1 (1986), pp. 59–71.

[32] F. Levi, *Die Teilung der projektiven Ebene durch Gerade oder Pseudogerade*, Ber. Math.-Phys. Kl. sächs. Akad. Wiss. Leipzig, 78 (1926), pp. 256–267.

[33] A. Marcus and G. Tardos, *Intersection reverse sequences and geometric applications*, in Proceedings of the 12th International Symposium on Graph Drawing, New York, NY, DIMACS, 2004.

[34] J. Matoušek, *Reporting points in halfspaces*, Comput. Geom., 2 (1992), pp. 169–186.

[35] J. Matoušek, *Range searching with efficient hierarchical cuttings*, Discrete Comput. Geom., 10 (1993), pp. 157–182.

[36] J. Matoušek, J. Pach, M. Sharir, S. Sifrony, and E. Welzl, *Fat triangles determine linearly many holes*, SIAM J. Comput., 23 (1994), pp. 154–169.

[37] K. Mehlhorn, *Data Structures and Algorithms* 3*: Multidimensional Searching and Computational Geometry*, EATCS Monographs on Theoretical Computer Science 3, Springer-Verlag, Heidelberg, Germany, 1984.

[38] M. Sharir and P. K. Agarwal, *Davenport-Schinzel Sequences and Their Geometric Applications*, Cambridge University Press, New York, 1995.

[39] M. Sharir and S. Smorodinsky, *Extremal configurations and levels in pseudoline arrangements*, in Algorithms and Data Structures, Lecture Notes in Comput. Sci. 2748, Springer-Verlag, Berlin, 2003, pp. 127–139.

[40] A. Solan, *Cutting cycles of rods in space*, in Proceedings of the 14th Annual ACM Symposium on Computational Geometry, Minneapolis, MN, 1998, pp. 135–142.

[41] L. Székely, *Crossing numbers and hard Erdös problems in discrete geometry*, Combin. Probab. Comput., 6 (1997), pp. 353–358.

[42] H. Tamaki and T. Tokuyama, *How to cut pseudo-parabolas into segments*, Discrete Comput. Geom., 19 (1998), pp. 265–290.

[43] H. Tamaki and T. Tokuyama, *A characterization of planar graphs by pseudo-line arrangements*, Algorithmica, 35 (2003), pp. 269–285.

SIAM J. Comput. © 2005 Society for Industrial and Applied Mathematics
Vol. 34, No. 3, pp. 553–579

# LAYOUT OF GRAPHS WITH BOUNDED TREE-WIDTH[*]

VIDA DUJMOVIĆ[†], PAT MORIN[‡], AND DAVID R. WOOD[†]

**Abstract.** A *queue layout* of a graph consists of a total order of the vertices, and a partition of the edges into *queues*, such that no two edges in the same queue are nested. The minimum number of queues in a queue layout of a graph is its *queue-number*. A *three-dimensional* (*straight-line grid*) *drawing* of a graph represents the vertices by points in $\mathbb{Z}^3$ and the edges by noncrossing line-segments. This paper contributes three main results:

(1) It is proved that the minimum volume of a certain type of three-dimensional drawing of a graph $G$ is closely related to the queue-number of $G$. In particular, if $G$ is an $n$-vertex member of a proper minor-closed family of graphs (such as a planar graph), then $G$ has a $\mathcal{O}(1) \times \mathcal{O}(1) \times \mathcal{O}(n)$ drawing if and only if $G$ has a $\mathcal{O}(1)$ queue-number.

(2) It is proved that the queue-number is bounded by the tree-width, thus resolving an open problem due to Ganley and Heath [*Discrete Appl. Math.*, 109 (2001), pp. 215–221] and disproving a conjecture of Pemmaraju [*Exploring the Powers of Stacks and Queues via Graph Layouts*, Ph. D. thesis, Virginia Polytechnic Institute and State University, Blacksburg, VA, 1992]. This result provides renewed hope for the positive resolution of a number of open problems in the theory of queue layouts.

(3) It is proved that graphs of bounded tree-width have three-dimensional drawings with $\mathcal{O}(n)$ volume. This is the most general family of graphs known to admit three-dimensional drawings with $\mathcal{O}(n)$ volume.

The proofs depend upon our results regarding *track layouts* and *tree-partitions* of graphs, which may be of independent interest.

**Key words.** queue layout, queue-number, three-dimensional graph drawing, tree-partition, tree-partition-width, tree-width, $k$-tree, track layout, track-number, acyclic coloring, acyclic chromatic number

**AMS subject classification.** 05C62

**DOI.** 10.1137/S0097539702416141

## 1. Introduction.

A *queue layout* of a graph consists of a total order of the vertices, and a partition of the edges into *queues*, such that no two edges in the same queue are nested. The dual concept of a *stack layout*, introduced by Ollmann [71] and commonly called a *book embedding*, is defined similarly, except that no two edges in the same *stack* may cross. The minimum number of queues (respectively, stacks) in a queue (stack) layout of a graph is its *queue-number* (*stack-number*). Queue layouts have been extensively studied [41, 53, 54, 58, 74, 78, 84, 86] with applications in parallel process scheduling, fault-tolerant processing, matrix computations, and sorting networks (see [74] for a survey). Queue layouts of directed acyclic graphs [9, 56, 57, 74] and posets [55, 74] have also been investigated. Our motivation for studying queue layouts is a connection with three-dimensional graph drawing.

Graph drawing is concerned with the automatic generation of aesthetically pleasing geometric representations of graphs. Graph drawing in the plane is well studied

[*]Received by the editors October 14, 2002; accepted for publication (in revised form) July 30, 2004; published electronically March 17, 2005. Results in this paper were presented previously at the GD '02 [33], FST TCS '02 [91], and WG '03 [36] conferences.
http://www.siam.org/journals/sicomp/34-3/41614.html

[†]School of Computer Science, McGill University, Montréal, QC, Canada H3A 2A7 (vida@cs.mcgill.ca).

[‡]School of Computer Science, Carleton University, Ottawa, ON, Canada K1S 5B6 (morin@scs.carleton.ca, davidw@scs.carleton.ca). The third author's research was supported by COMBSTRU and by NSERC.

(see [23, 64]). Motivated by experimental evidence suggesting that displaying a graph in three dimensions is better than in two [88, 89], and applications including information visualisation [88], VLSI circuit design [66], and software engineering [90], there is a growing body of research in three-dimensional graph drawing. In this paper we study *three-dimensional straight-line grid drawings*, or *three-dimensional drawings* for short. In this model, vertices are positioned at grid-points in $\mathbb{Z}^3$, and edges are drawn as straight line-segments with no crossings [16, 20, 24, 26, 27, 42, 53, 76, 73]. We focus on the problem of producing three-dimensional drawings with small volume. Three-dimensional drawings with vertices in $\mathbb{R}^3$ have also been studied [39, 47, 18, 15, 17, 61, 21, 63, 60, 62, 68, 72]. Aesthetic criteria besides volume that have been considered include symmetry [60, 61, 62, 63], aspect ratio [18, 47], angular resolution [47, 18], edge-separation [18, 47], and convexity [17, 18, 39, 85].

The first main result of this paper (Theorem 2.10) reduces the question of whether a graph has a three-dimensional drawing with small volume to a question regarding queue layouts. In particular, we prove that every $n$-vertex graph from a proper minor-closed graph family $\mathcal{G}$ has a $\mathcal{O}(1) \times \mathcal{O}(1) \times \mathcal{O}(n)$ drawing if and only if $\mathcal{G}$ has a $\mathcal{O}(1)$ queue-number, and this result holds true when replacing $\mathcal{O}(1)$ by $\mathcal{O}(\operatorname{polylog} n)$. Consider the family of planar graphs, which are minor-closed. (In the conference version of their paper) Felsner, Liotta, and Wismath [42] asked whether every planar graph has a three-dimensional drawing with $\mathcal{O}(n)$ volume. Heath and Rosenberg [58] and Heath Leighton, and Rosenberg [54] asked whether every planar graph has a $\mathcal{O}(1)$ queue-number. By our result, these two open problems are almost equivalent in the following sense. If every planar graph has $\mathcal{O}(1)$ queue-number, then every planar graph has a three-dimensional drawing with $\mathcal{O}(n)$ volume. Conversely, if every planar graph has a $\mathcal{O}(1) \times \mathcal{O}(1) \times \mathcal{O}(n)$ drawing, then every planar graph has $\mathcal{O}(1)$ queue-number. It is possible, however, that planar graphs have unbounded queue-number, yet have, say, $\mathcal{O}(n^{1/3}) \times \mathcal{O}(n^{1/3}) \times \mathcal{O}(n^{1/3})$ drawings.

Our other main results regard three-dimensional drawings and queue layouts of graphs with bounded tree-width. Tree-width, first defined by Halin [50], although largely unnoticed until independently rediscovered by Robertson and Seymour [79] and Arnborg and Proskurowski [7], is a measure of the similarity of a graph to a tree (see section 2.1 for the definition). Tree-width (or its special case, path-width) has been previously used in the context of graph drawing by Dujmović et al. [32], Hliněný [59], and Peng [75], for example.

The second main result (Corollary 2.8) is that the queue-number of a graph is bounded by its tree-width. This solves an open problem due to Ganley and Heath [45], who proved that stack-number is bounded by tree-width and asked whether a similar relationship holds for queue-number. This result has significant implications for the above open problem (does every planar graph have $\mathcal{O}(1)$ queue-number), and the more general question (since planar graphs have stack-number at most four [93]) of whether queue-number is bounded by stack-number. Heath and colleagues [58, 54] originally conjectured that both of these questions have an affirmative answer. More recently, however, Pemmaraju [74] conjectured that the "stellated $K_3$," a planar 3-tree, has $\Theta(\log n)$ queue-number, and provided evidence to support this conjecture (also see [45]). This suggested that the answer to both of the above questions was negative. In particular, Pemmaraju [74] and Heath [private communication, 2002] conjectured that planar graphs have $\mathcal{O}(\log n)$ queue-number. However, our result provides a queue layout of *any* 3-tree, and thus the stellated $K_3$, with $\mathcal{O}(1)$ queues. Hence our result disproves the first conjecture of Pemmaraju [74] mentioned above and renews hope in an affirmative answer to the above open problems.

The third main result is that every graph of bounded tree-width has a three-dimensional drawing with $\mathcal{O}(n)$ volume. The family of graphs of bounded tree-width includes most of the graphs previously known to admit three-dimensional drawings with $\mathcal{O}(n)$ volume (for example, outerplanar graphs), and also includes many graph families for which the previous best volume bound was $\mathcal{O}(n^2)$ (for example, series-parallel graphs). Many graphs arising in applications of graph drawing do have small tree-width. Outerplanar and series-parallel graphs are the obvious examples. Another example arises in software engineering applications. Thorup [87] proved that the control-flow graphs of go-to free programs in many programming languages have tree-width bounded by a small constant; in particular, 3 for Pascal and 6 for C. Other families of graphs having bounded tree-width (for constant $k$) include almost trees with parameter $k$, graphs with a feedback vertex set of size $k$, band-width $k$ graphs, cut-width $k$ graphs, planar graphs of radius $k$, and $k$-outerplanar graphs. If the size of a maximum clique is a constant $k$, then chordal, interval, and circular arc graphs also have bounded tree-width. Thus, by our result, all of these graphs have three-dimensional drawings with $\mathcal{O}(n)$ volume, and $\mathcal{O}(1)$ queue-number.

To prove our results for graphs of bounded tree-width, we employ a related structure called a tree-partition, introduced independently by Seese [83] and Halin [51]. A *tree-partition* of a graph is a partition of its vertices into "bags" such that contracting each bag to a single vertex gives a forest (after deleting loops and replacing parallel edges by a single edge). In a result of independent interest, we prove that every $k$-tree has a tree-partition such that each bag induces a connected $(k-1)$-tree, amongst other properties. The second tool that we use is a *track layout*, which consists of a vertex-coloring and a total order of each color class, such that between any two color classes no two edges cross.

The remainder of the paper is organized as follows. In section 2 we introduce the required background material, state our results regarding three-dimensional drawings and queue layouts, and compare these with results in the literature. In section 3 we establish a number of results concerning track layouts. That three-dimensional drawings and queue-layouts are closely related stems from the fact that three-dimensional drawings and queue layouts are both closely related to track layouts, as proved in section 4 and section 5, respectively. In section 6 we prove the above-mentioned theorem for tree-partitions of $k$-trees, which is used in section 7 to construct track layouts of graphs with bounded tree-width. We conclude in section 8 with a number of open problems.

## 2. Background and results.

Throughout this paper all graphs $G$ are undirected, simple, and finite with vertex set $V(G)$ and edge set $E(G)$. The number of vertices and the maximum degree of $G$ are respectively denoted by $n = |V(G)|$ and $\Delta(G)$. The subgraph induced by a set of vertices $A \subseteq V(G)$ is denoted by $G[A]$. For all disjoint subsets $A, B \subseteq V(G)$, the bipartite subgraph of $G$ with vertex set $A \cup B$ and edge set $\{vw \in E(G) : v \in A, w \in B\}$ is denoted by $G[A, B]$.

A graph $H$ is a *minor* of a graph $G$ if $H$ is isomorphic to a graph obtained from a subgraph of $G$ by contracting edges. A family of graphs closed under taking minors is *proper* if it is not the class of all graphs.

A *graph parameter* is a function $\alpha$ that assigns to every graph $G$ a nonnegative integer $\alpha(G)$. Let $\mathcal{G}$ be a family of graphs. By $\alpha(\mathcal{G})$ we denote the function $f : \mathbb{N} \to \mathbb{N}$, where $f(n)$ is the maximum of $\alpha(G)$ taken over all $n$-vertex graphs $G \in \mathcal{G}$. We say that $\mathcal{G}$ has *bounded* $\alpha$ if $\alpha(\mathcal{G}) \in \mathcal{O}(1)$. A graph parameter $\alpha$ is *bounded by* a graph parameter $\beta$ (for some graph family $\mathcal{G}$), if there exists a function $g$ such that $\alpha(G) \leq g(\beta(G))$ for every graph $G$ (in $\mathcal{G}$).

**2.1. Tree-width.** Let $G$ be a graph and let $T$ be a tree. An element of $V(T)$ is called a *node*. Let $\{T_x \subseteq V(G) : x \in V(T)\}$ be a set of subsets of $V(G)$ indexed by the nodes of $T$. Each $T_x$ is called a *bag*. The pair $(T, \{T_x : x \in V(T)\})$ is a *tree-decomposition* of $G$ if

1. $\bigcup_{x \in V(T)} T_x = V(G)$ (that is, every vertex of $G$ is in at least one bag),
2. $\forall$ edges $vw$ of $G$, $\exists$ node $x$ of $T$ such that $v \in T_x$ and $w \in T_x$, and
3. $\forall$ nodes $x, y, z$ of $T$, if $y$ is on the path from $x$ to $z$ in $T$, then $T_x \cap T_z \subseteq T_y$.

The *width* of a tree-decomposition is one less than the maximum cardinality of a bag. A *path-decomposition* is a tree-decomposition where the tree $T$ is a path $T = (x_1, x_2, \dots, x_m)$, which is simply identified by the sequence of bags $T_1, T_2, \dots, T_m$ where each $T_i = T_{x_i}$. The *path-width* (respectively, *tree-width*) of a graph $G$, denoted by $\mathsf{pw}(G)$ ($\mathsf{tw}(G)$), is the minimum width of a path- (tree-) decomposition of $G$. Graphs with tree-width at most one are precisely the forests. Graphs with tree-width at most two are called *series-parallel*,[1] and are characterized as those graphs with no $K_4$ minor (see [10]).

A *k-tree* for some $k \in \mathbb{N}$ is defined recursively as follows. The empty graph is a $k$-tree, and the graph obtained from a $k$-tree by adding a new vertex adjacent to each vertex of a clique with at most $k$ vertices is also a $k$-tree. This definition of a $k$-tree is by Reed [77]. The following more restrictive definition of a $k$-tree, which we call "strict," was introduced by Arnborg and Proskurowski [7], and is more often used in the literature. A $k$-clique is a *strict k-tree*, and the graph obtained from a strict $k$-tree by adding a new vertex adjacent to each vertex of a $k$-clique is also a strict $k$-tree. Obviously the strict $k$-trees are a proper subclass of the $k$-trees. A subgraph of a $k$-tree is called a *partial k-tree*, and a subgraph of a strict $k$-tree is called a *partial strict k-tree*. The following result is well known (see, for example, [10, 77]). A *chord* of a cycle $C$ is an edge not in $C$ whose end-vertices are both in $C$. A graph is *chordal* if every cycle on at least four vertices has a chord.

LEMMA 2.1. *Let $G$ be a graph. The following are equivalent:*

1. *$G$ has tree-width $\mathsf{tw}(G) \leq k$,*
2. *$G$ is a partial $k$-tree,*
3. *$G$ is a partial strict $k$-tree,*
4. *$G$ is a subgraph of a chordal graph that has no clique on $k + 2$ vertices.*

*Proof.* Scheffler [81] proved that (1) and (3) are equivalent. That (1) and (4) are equivalent is due to Robertson and Seymour [79]. That (2) and (4) are equivalent is the characterization of chordal graphs in terms of "perfect elimination" vertex-orderings due to Fulkerson and Gross [44]. $\square$

**2.2. Tree-partitions.** As in the definition of a tree-decomposition, let $G$ be a graph and let $\{T_x \subseteq V(G) : x \in V(T)\}$ be a set of subsets of $V(G)$ (called *bags*) indexed by the nodes of a tree $T$. The pair $(T, \{T_x : x \in V(T)\})$ is a *tree-partition* of $G$ if

1. $\forall$ distinct nodes $x$ and $y$ of $T$, $T_x \cap T_y = \emptyset$, and
2. $\forall$ edges $vw$ of $G$, either
   (i) $\exists$ node $x$ of $T$ with $v \in T_x$ and $w \in T_x$ ($vw$ is called an *intrabag* edge), or

[1] "Series-parallel digraphs" are often defined in terms of certain "series" and "parallel" composition operations. The underlying undirected graph of such a digraph has tree-width at most two (see [10]).

(ii) $\exists$ edge $xy$ of $T$ with $v \in T_x$ and $w \in T_y$ ($vw$ is called an *interbag* edge).

The main property of tree-partitions that has been studied in the literature is the maximum cardinality of a bag, called the *width* of the tree-partition [11, 51, 83, 30, 31]. The minimum width over all tree-partitions of a graph $G$ is the *tree-partition-width*[2] of $G$, denoted by $\mathsf{tpw}(G)$. A graph with bounded degree has bounded tree-partition-width if and only if it has bounded tree-width [31]. In particular, for every graph $G$, Ding and Oporowski [30] proved that $\mathsf{tpw}(G) \leq 24\,\mathsf{tw}(G)\,\Delta(G)$ (assuming $\Delta(G) \geq 1$), and Seese [83] proved that $\mathsf{tw}(G) \leq 2\,\mathsf{tpw}(G) - 1$.

Theorem 6.1 provides a tree-partition of a $k$-tree $G$ with additional features besides small width. First, the subgraph induced by each bag is a connected $(k-1)$-tree. This allows us to perform induction on $k$. Second, in each nonroot bag $T_x$ the set of vertices in the parent bag of $x$ with a neighbor in $T_x$ form a clique. This feature is crucial in the intended application (Theorem 7.3). Finally the tree-partition has width at most $\max\{1, k(\Delta(G) - 1)\}$, which represents a constant-factor improvement over the above result by Ding and Oporowski [30] in the case of $k$-trees.

**2.3. Track layouts.** Let $G$ be a graph. A *coloring* of $G$ is a partition $\{V_i : i \in I\}$ of $V(G)$, where $I$ is a set of *colors*, such that for every edge $vw$ of $G$, if $v \in V_i$ and $w \in V_j$, then $i \neq j$. Each set $V_i$ is called a *color class*. A coloring of $G$ with $c$ colors is a *$c$-coloring*, and we say that $G$ is *$c$-colorable*. The *chromatic number* of $G$, denoted by $\chi(G)$, is the minimum $c$ such that $G$ is $c$-colorable.

If $<_i$ is a total order of a color class $V_i$, then we call the pair $(V_i, <_i)$ a *track*. If $\{V_i : i \in I\}$ is a coloring of $G$ and $(V_i, <_i)$ is a track for each color $i \in I$, then we say $\{(V_i, <_i) : i \in I\}$ is a *track assignment* of $G$ *indexed by* $I$. Note that at times it will be convenient to also refer to a color $i \in I$ and the color class $V_i$ as a *track*. The precise meaning will always be clear from the context. A *$t$-track assignment* is a track assignment with $t$ tracks.

As illustrated in Figure 2.1, an *X-crossing* in a track assignment consists of two edges $vw$ and $xy$ such that $v <_i x$ and $y <_j w$ for distinct tracks $V_i$ and $V_j$. A $t$-track assignment with no X-crossing is called a *$t$-track layout*. The *track-number* of a graph $G$, denoted by $\mathsf{tn}(G)$, is the minimum $t$ such that $G$ has a $t$-track layout.

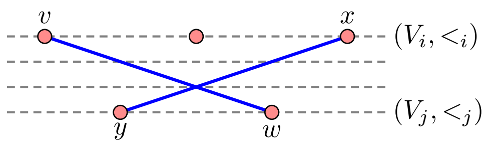


FIG. 2.1. *An example of an X-crossing in a track assignment.*

Let $\{(V_i, <_i) : i \in I\}$ be a $t$-track layout of a graph $G$. The *span* of an edge $vw$ of $G$, with respect to a numbering of the tracks $I = \{1, 2, \ldots, t\}$, is defined to be $|i - j|$, where $v \in V_i$ and $w \in V_j$.

Track layouts will be central in most of our proofs. To enable comparison of our results to those in the literature we now introduce the notion of an "improper" track layout. A *improper coloring* of a graph $G$ is simply a partition $\{V_i : i \in I\}$ of $V(G)$. Here adjacent vertices may be in the same color class. A track of an improper coloring is defined as above. Suppose $\{V_i : i \in I\}$ is an improper coloring of $G$ and $(V_i, <_i)$ is a track for each color $i \in I$. An edge with both end-vertices in the same

[2]Tree-partition-width has also been called *strong tree-width* [83, 11].

track is called an *intratrack* edge; otherwise it is called an *intertrack* edge. We say that $\{(V_i, <_i) : i \in I\}$ is an *improper track assignment* of $G$ if, for all intratrack edges $vw \in E(G)$ with $v \in V_i$ and $w \in V_i$ for some $i \in I$, there is no vertex $x$ with $v <_i x <_i w$. That is, adjacent vertices in the same track are consecutive in that track. An improper $t$-track assignment with no X-crossing is called an *improper $t$-track layout.*[3]

LEMMA 2.2. *If a graph $G$ has an improper $t$-track layout, then $G$ has a $2t$-track layout.*

*Proof.* For every track $V_i$ of an improper $t$-track layout of $G$, let $V_i'$ be a new track. Move every second vertex from $V_i$ to $V_i'$ such that $V_i'$ inherits its total order from the original $V_i$. Clearly there is no intratrack edge and no X-crossing. Thus we obtain a $2t$-track layout of $G$. $\square$

Hence the track-number of a graph is at most twice its "improper track-number." The following lemma, which was jointly discovered with Giuseppe Liotta, gives a compelling reason to only consider proper track layouts. Similar ideas can be found in [42, 26]. Let $vw$ be an edge of a graph $G$. Let $G'$ be the graph obtained from $G$ by adding a new vertex $x$ only adjacent to $v$ and $w$. We say $x$ is an *ear*, and $G'$ is obtained from $G$ by *adding an ear to $vw$*.

LEMMA 2.3. *Let $\mathcal{G}$ be a class of graphs closed under the addition of ears (for example, series-parallel graphs or planar graphs). If every graph in $\mathcal{G}$ has an improper $t$-track layout for some constant $t$, then every graph in $\mathcal{G}$ has a (proper) $t$-track layout.*

*Proof.* For any graph $G \in \mathcal{G}$, let $G'$ be the graph obtained from $G$ by adding $t$ ears to every edge of $G$. By assumption, $G'$ has an improper $t$-track layout. Suppose that there is an edge $vw$ of $G$ such that $v$ and $w$ are in the same track. None of the ears added to $vw$ are on the same track, as otherwise adjacent vertices would not be consecutive in that track. Thus there is a track containing at least two of the ears added to $vw$. However, this implies that there is an X-crossing, which is a contradiction. Thus the end-vertices of every edge of $G$ are in distinct tracks. Hence the improper $t$-track layout of $G'$ contains a $t$-track layout of $G$. $\square$

Lemmas 2.2 and 2.3 imply that only for relatively small classes of graphs will the distinction between track layouts and improper track layouts be significant. We therefore chose to work with the less cumbersome notion of a track layout. The following theorem summarizes our bounds on the track-number of a graph.

THEOREM 2.4. *Let $G$ be a graph with maximum degree $\Delta(G)$, path-width $\mathsf{pw}(G)$, tree-partition-width $\mathsf{tpw}(G)$, and tree-width $\mathsf{tw}(G)$. The track-number of $G$ satisfies*

(a) $\mathsf{tn}(G) \leq \mathsf{pw}(G) + 1 \leq 1 + (\mathsf{tw}(G) + 1)\log n$,

(b) $\mathsf{tn}(G) \leq 3\,\mathsf{tpw}(G) \leq 72\,\mathsf{tw}(G)\,\Delta(G)$ (*assuming* $\Delta(G) \geq 1$),

(c) $\mathsf{tn}(G) \leq 3^{\mathsf{tw}(G)} \cdot 6^{(4^{\mathsf{tw}(G)} - 3\,\mathsf{tw}(G) - 1)/9}$.

*Proof.* Part (a) follows from Lemma 3.2 and the fact that $\mathsf{pw}(G) \leq (\mathsf{tw}(G) + 1)\log n$ (see [10]). Note that $\mathsf{tn}(G) \leq 1 + (\mathsf{tw}(G)+1)\log n$ can be proved directly using a separator-based approach similar to that used to prove $\mathsf{pw}(G) \leq (\mathsf{tw}(G) + 1)\log n$. Part (b) follows from Lemma 3.3 in section 3 and the result of Ding and Oporowski [30] discussed in section 2.2. Part (c) is Theorem 7.3. $\square$

## 2.4. Vertex-orderings.

Let $G$ be a graph. A total order $\sigma = (v_1, v_2, \ldots, v_n)$ of $V(G)$ is called a *vertex-ordering* of $G$. Suppose that $G$ is connected. The *depth* of a vertex $v_i$ in $\sigma$ is the graph-theoretic distance between $v_1$ and $v_i$ in $G$. We say that

[3]In [33, 35, 91] we called a track layout an *ordered layering with no X-crossing and no intralayer edges*, and an improper track layout was called an *ordered layering with no X-crossing.*

$\sigma$ is a *breadth-first* vertex-ordering if for all vertices $v$ and $w$ with $v <_\sigma w$ the depth of $v$ in $\sigma$ is no more than the depth of $w$ in $\sigma$. Vertex-orderings, and in particular, vertex-orderings of trees, will be used extensively in this paper. Consider a breadth-first vertex-ordering $\sigma$ of a tree $T$ such that vertices at depth $d \geq 1$ are ordered with respect to the ordering of vertices at depth $d-1$. In particular, if $v$ and $x$ are vertices at depth $d$ with respective parents $w$ and $y$ at depth $d-1$ with $w <_\sigma y$, then $v <_\sigma x$. Such a vertex-ordering is called a *lexicographical* breadth-first vertex-ordering of $T$, and is illustrated in Figure 2.2.

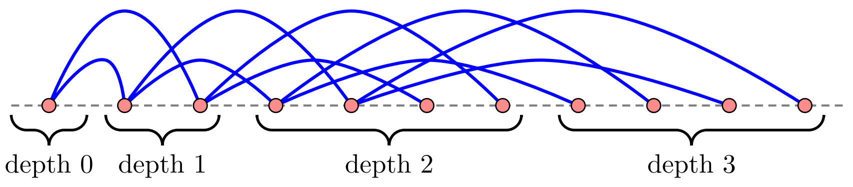


FIG. 2.2. *A lexicographical breadth-first vertex-ordering of a tree.*

**2.5. Queue layouts.** A *queue layout* of a graph $G$ consists of a vertex-ordering $\sigma$ of $G$ and a partition of $E(G)$ into *queues* such that no two edges in the same queue are *nested* with respect to $\sigma$. That is, there are no edges $vw$ and $xy$ in a single queue with $v <_\sigma x <_\sigma y <_\sigma w$. The minimum number of queues in a queue layout of $G$ is called the *queue-number* of $G$ and is denoted by $\mathsf{qn}(G)$. A similar concept is that of a *stack layout* (or *book embedding*), which consists of a vertex-ordering $\sigma$ of $G$ and a partition of $E(G)$ into *stacks* (or *pages*) such that there are no edges $vw$ and $xy$ in a single stack with $v <_\sigma x <_\sigma w <_\sigma y$. The minimum number of stacks in a stack layout of $G$ is called the *stack-number* (or *page-number* or *book-thickness*) of $G$ and is denoted by $\mathsf{sn}(G)$. A queue (respectively, stack) layout with $k$ queues (stacks) is called a *$k$-queue* (*$k$-stack*) *layout*, and a graph that admits a $k$-queue ($k$-stack) layout is called a *$k$-queue* (*$k$-stack*) *graph*.

Heath and Rosenberg [58] characterized 1-queue graphs as the "arched levelled planar" graphs, and proved that it is $\mathcal{NP}$-complete to recognize such graphs. This result is in contrast to the situation for stack layouts—1-stack graphs are precisely the outerplanar graphs [8], which can be recognized in polynomial time. Heath, Leighton, and Rosenberg [54] proved that 1-stack graphs are 2-queue graphs (rediscovered by Rengarajan and Veni Madhavan [78]), and that 1-queue graphs are 2-stack graphs.

While it is $\mathcal{NP}$-hard to minimize the number of stacks in a stack layout given a fixed vertex-ordering [46], the analogous problem for queue layouts can be solved as follows. A *$k$-rainbow* in a vertex-ordering $\sigma$ consists of a matching $\{v_i w_i : 1 \leq i \leq k\}$ such that $v_1 <_\sigma v_2 <_\sigma \cdots <_\sigma v_k <_\sigma w_k <_\sigma w_{k-1} <_\sigma \cdots <_\sigma w_1$, as illustrated in Figure 2.3.

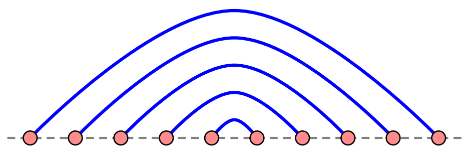


FIG. 2.3. *A rainbow of five edges in a vertex-ordering.*

A vertex-ordering containing a $k$-rainbow needs at least $k$ queues. A straightforward application of Dilworth's Theorem [29] proves the converse. That is, a fixed vertex-ordering admits a $k$-queue layout, where $k$ is the size of the largest rainbow. (Heath and Rosenberg [58] describe a $\mathcal{O}(m \log\log n)$ time algorithm to compute the queue assignment.) Thus determining $\mathsf{qn}(G)$ can be viewed as the following vertex-ordering problem.

LEMMA 2.5 (see [58]). *The queue-number* $\mathsf{qn}(G)$ *of a graph* $G$ *is the minimum, taken over all vertex-orderings* $\sigma$ *of* $G$*, of the maximum size of a rainbow in* $\sigma$. □

Stack and/or queue layouts of $k$-trees have previously been investigated in [19, 78, 45]. A 1-tree is a 1-queue graph, since in a lexicographical breadth-first vertex-ordering of a tree no two edges are nested (see Figure 2.2). Chung, Leighton, and Rosenberg [19] proved that in a depth-first vertex-ordering of a tree no two edges cross. Thus 1-trees are 1-stack graphs. Rengarajan and Veni Madhavan [78] proved that graphs with tree-width at most two (the series parallel graphs) are 2-stack and 3-queue graphs.[4] Improper track layouts are implicit in the work of Heath, Leighton, and Rosenberg [54] and Rengarajan and Veni Madhavan [78]. In section 5 we prove the following fundamental relationship between queue and track layouts.

THEOREM 2.6. *For every graph* $G$*,* $\mathsf{qn}(G) \leq \mathsf{tn}(G) - 1$*. Moreover, if* $\mathcal{G}$ *is any proper minor-closed graph family, then* $\mathcal{G}$ *has queue-number* $\mathsf{qn}(\mathcal{G}) \in \mathcal{F}(n)$ *if and only if* $\mathcal{G}$ *has track-number* $\mathsf{tn}(\mathcal{G}) \in \mathcal{F}(n)$*, where* $\mathcal{F}(n)$ *is any family of functions closed under multiplication* (*such as* $\mathcal{O}(1)$ *or* $\mathcal{O}(\operatorname{polylog} n)$).

Ganley and Heath [45] proved that every graph $G$ has stack-number $\mathsf{sn}(G) \leq \mathsf{tw}(G) + 1$ (using a depth-first traversal of a tree-decomposition), and asked whether queue-number is bounded by tree-width. One of the principal results of this paper is to solve this question in the affirmative. Applying Theorems 2.4 and 2.6, we have the following.

THEOREM 2.7. *Let* $G$ *be a graph with maximum degree* $\Delta(G)$*, path-width* $\mathsf{pw}(G)$*, tree-partition-width* $\mathsf{tpw}(G)$*, and tree-width* $\mathsf{tw}(G)$*. The queue-number* $\mathsf{qn}(G)$ *satisfies*[5]

(a) $\mathsf{qn}(G) \leq \mathsf{pw}(G) \leq (\mathsf{tw}(G)+1)\log n$,

(b) $\mathsf{qn}(G) \leq 3\,\mathsf{tpw}(G) - 1 \leq 72\,\mathsf{tw}(G)\,\Delta(G) - 1$ (*assuming* $\Delta(G) \geq 1$),

(c) $\mathsf{qn}(G) \leq 3^{\mathsf{tw}(G)} \cdot 6^{(4^{\mathsf{tw}(G)} - 3\,\mathsf{tw}(G) - 1)/9} - 1$. □

A similar upper bound to Theorem 2.7(a) was obtained by Heath and Rosenberg [58], who proved that every graph $G$ has $\mathsf{qn}(G) \leq \lceil \frac{1}{2}\mathsf{bw}(G) \rceil$, where $\mathsf{bw}(G)$ is the band-width of $G$. In many cases this result is weaker than Theorem 2.7(a) since $\mathsf{pw}(G) \leq \mathsf{bw}(G)$ (see [28]). More importantly, we have the following corollary of Theorem 2.7(c).

COROLLARY 2.8. *Queue-number is bounded by tree-width, and hence graphs with bounded tree-width have bounded queue-number.* □

**2.6. Three-dimensional drawings.** A *three-dimensional straight-line grid drawing* of a graph, henceforth called a *three-dimensional drawing*, represents the vertices by distinct points in $\mathbb{Z}^3$ (called *grid-points*) and represents each edge as a line-segment between its end-vertices, such that edges intersect only at common end-vertices, and an edge only intersects a vertex that is an end-vertex of that edge.

In contrast to the case in the plane, a folklore result states that every graph has a three-dimensional drawing. Such a drawing can be constructed using the "moment

[4]In [35] we give a simple proof based on Theorem 6.1 for the result by Rengarajan and Veni Madhavan [78] that every series-parallel graph has a 3-queue layout.

[5]In [91] we obtained an alternative proof that $\mathsf{qn}(G) \leq \mathsf{pw}(G)$ using the "vertex separation number" of a graph (which equals its path-width); applying Lemma 2.5 directly, we proved that $\mathsf{qn}(G) \leq \frac{3}{2}\,\mathsf{tpw}(G)$, and thus $\mathsf{qn}(G) \leq 36\,\Delta(G)\,\mathsf{tw}(G)$.

curve" algorithm in which vertex $v_i$, $1 \leq i \leq n$, is represented by the grid-point $(i, i^2, i^3)$. It is easily seen—compare with Lemma 4.2—that no two edges cross. (Two edges *cross* if they intersect at some point other than a common end-vertex.)

Since every graph has a three-dimensional drawing, we are interested in optimizing certain measures of the aesthetic quality of a drawing. If a three-dimensional drawing is contained in an axis-aligned box with side lengths $X-1$, $Y-1$, and $Z-1$, then we speak of an $X \times Y \times Z$ drawing with *volume* $X \cdot Y \cdot Z$ and *aspect ratio* $\max\{X,Y,Z\}/\min\{X,Y,Z\}$. This paper considers the problem of producing a three-dimensional drawing of a given graph with small volume, and with small aspect ratio as a secondary criterion.

Observe that the drawings produced by the moment curve algorithm have $\mathcal{O}(n^6)$ volume. Cohen et al. [20] improved this bound by proving that if $p$ is a prime with $n < p \leq 2n$, and each vertex $v_i$ is represented by the grid-point $(i, i^2 \bmod p, i^3 \bmod p)$, then there is still no crossing. This construction is a generalization of an analogous two-dimensional technique due to Erdős [40]. Furthermore, Cohen et al. [20] proved that the resulting $\mathcal{O}(n^3)$ volume bound is asymptotically optimal in the case of the complete graph $K_n$. It is therefore of interest to identify fixed graph parameters that allow for three-dimensional drawings with small volume.

The first such parameter to be studied was the chromatic number [16, 73]. Calamoneri and Sterbini [16] proved that every 4-colorable graph has a three-dimensional drawing with $\mathcal{O}(n^2)$ volume. Generalizing this result, Pach, Thiele, and Tóth [73] proved that graphs of bounded chromatic number have three-dimensional drawings with $\mathcal{O}(n^2)$ volume, and that this bound is asymptotically optimal for the complete bipartite graph with equal sized bipartitions. If $p$ is a suitably chosen prime, the main step of this algorithm represents the vertices in the $i$th color class by grid-points in the set $\{(i, t, it) : t \equiv i^2 \pmod{p}\}$. It follows that the volume bound is $\mathcal{O}(k^2 n^2)$ for $k$-colorable graphs.

The lower bound of Pach, Thiele, and Tóth [73] for the complete bipartite graph was generalized by Bose et al. [14] for all graphs. They proved that every three-dimensional drawing with $n$ vertices and $m$ edges has volume at least $\frac{1}{8}(n+m)$. In particular, the maximum number of edges in an $X \times Y \times Z$ drawing is exactly $(2X-1)(2Y-1)(2Z-1) - XYZ$. For example, graphs admitting three-dimensional drawings with $\mathcal{O}(n)$ volume have $\mathcal{O}(n)$ edges.

The first nontrivial $\mathcal{O}(n)$ volume bound was established by Felsner, Liotta, and Wismath [42] for outerplanar graphs. Their elegant algorithm "wraps" a two-dimensional drawing around a triangular prism to obtain an improper 3-track layout (see Lemmas 3.1 and 3.4 for more on this method). Poranen [76] proved that series-parallel digraphs have upward three-dimensional drawings with $\mathcal{O}(n^3)$ volume, and that this bound can be improved to $\mathcal{O}(n^2)$ and $\mathcal{O}(n)$ in certain special cases. Di Giacomo, Liotta, and Wismath [26] proved that series-parallel graphs with maximum degree three have three-dimensional drawings with $\mathcal{O}(n)$ volume.

In section 4 we prove the following intrinsic relationship between three-dimensional drawings and track layouts.

THEOREM 2.9. *Every graph $G$ has a $\mathcal{O}(\mathsf{tn}(G)) \times \mathcal{O}(\mathsf{tn}(G)) \times \mathcal{O}(n)$ drawing. Moreover, $G$ has an $\mathcal{F}(n) \times \mathcal{F}(n) \times \mathcal{O}(n)$ drawing if and only if $G$ has track-number $\mathsf{tn}(G) \in \mathcal{F}(n)$, where $\mathcal{F}(n)$ is a family of functions closed under multiplication.*

Of course, every graph has an $n$-track layout—simply place a single vertex on each track. Thus Theorem 2.9 matches the $\mathcal{O}(n^3)$ volume bound discussed in section 2.6. In fact, the drawings of $K_n$ produced by our algorithm, with each vertex in a distinct track, are identical to those produced by the algorithm of Cohen et al. [20].

Theorems 2.6 and 2.9 immediately imply the following result, which reduces the problem of producing a three-dimensional drawing with small volume to that of producing a queue layout of the same graph with few queues.

THEOREM 2.10. *Let $\mathcal{G}$ be a proper minor-closed family of graphs, and let $\mathcal{F}(n)$ be a family of functions closed under multiplication. The following are equivalent:*

(a) *every $n$-vertex graph in $\mathcal{G}$ has an $\mathcal{F}(n) \times \mathcal{F}(n) \times \mathcal{O}(n)$ drawing,*

(b) *$\mathcal{G}$ has track-number $\mathsf{tn}(\mathcal{G}) \in \mathcal{F}(n)$, and*

(c) *$\mathcal{G}$ has queue-number $\mathsf{qn}(\mathcal{G}) \in \mathcal{F}(n)$.* □

Graphs with constant queue-number include de Bruijn graphs, FFT, and Beneš network graphs [58]. By Theorem 2.10, these graphs have three-dimensional drawings with $\mathcal{O}(n)$ volume. Applying Theorems 2.4 and 2.9, we have the following result.

THEOREM 2.11. *Let $G$ be a graph with maximum degree $\Delta(G)$, path-width $\mathsf{pw}(G)$, tree-partition-width $\mathsf{tpw}(G)$, and tree-width $\mathsf{tw}(G)$. Then $G$ has a three-dimensional drawing with the following dimensions:*

(a) $\mathcal{O}(\mathsf{pw}(G)) \times \mathcal{O}(\mathsf{pw}(G)) \times \mathcal{O}(n)$*, which is* $\mathcal{O}(\mathsf{tw}(G) \log n) \times \mathcal{O}(\mathsf{tw}(G) \log n) \times \mathcal{O}(n)$*,*

(b) $\mathcal{O}(\mathsf{tpw}(G)) \times \mathcal{O}(\mathsf{tpw}(G)) \times \mathcal{O}(n)$*, which is* $\mathcal{O}(\Delta(G)\,\mathsf{tw}(G)) \times \mathcal{O}(\Delta(G)\,\mathsf{tw}(G)) \times \mathcal{O}(n)$*,*

(c) $\mathcal{O}(3^{\mathsf{tw}(G)} \cdot 6^{(4^{\mathsf{tw}(G)} - 3\,\mathsf{tw}(G) - 1)/9}) \times \mathcal{O}(3^{\mathsf{tw}(G)} \cdot 6^{(4^{\mathsf{tw}(G)} - 3\,\mathsf{tw}(G) - 1)/9}) \times \mathcal{O}(n)$. □

Most importantly, we have the following corollary of Theorem 2.11(c).

COROLLARY 2.12. *Every graph with bounded tree-width has a three-dimensional drawing with $\mathcal{O}(n)$ volume.* □

Note that bounded tree-width is not necessary for a graph to have a three-dimensional drawing with $\mathcal{O}(n)$ volume. The $\sqrt{n} \times \sqrt{n}$ plane grid graph has $\Theta(\sqrt{n})$ tree-width, and has a $\sqrt{n} \times \sqrt{n} \times 1$ drawing with $n$ volume. It also has a 3-track layout, and thus, by Lemma 4.2, has a $\mathcal{O}(1) \times \mathcal{O}(1) \times \mathcal{O}(n)$ drawing.

Since a planar graph is 4-colorable, by the results of Calamoneri and Sterbini [16] and Pach, Thiele, and Tóth [73] discussed above, every planar graph has a three-dimensional drawing with $\mathcal{O}(n^2)$ volume. This result also follows from the classical algorithms of de Fraysseix, Pach, and Pollack [22] and Schnyder [82] for producing $\mathcal{O}(n) \times \mathcal{O}(n)$ plane grid drawings. All of these methods produce $\mathcal{O}(n) \times \mathcal{O}(n) \times \mathcal{O}(1)$ drawings, which have $\Theta(n)$ aspect ratio. Since every planar graph $G$ has $\mathsf{pw}(G) \in \mathcal{O}(\sqrt{n})$ [10], we have the following corollary of Theorem 2.11(a).

COROLLARY 2.13. *Every planar graph has a three-dimensional drawing with $\mathcal{O}(n^2)$ volume and $\Theta(\sqrt{n})$ aspect ratio.* □

This result matches the above $\mathcal{O}(n^2)$ volume bounds with an improvement in the aspect ratio by a factor of $\Theta(\sqrt{n})$. Our final result regarding three-dimensional drawings, which is proved in section 4, examines the apparent trade-off between aspect ratio and volume.

THEOREM 2.14. *For every graph $G$ and for every $r$, $1 \leq r \leq n/\mathsf{tn}(G)$, $G$ has a three-dimensional drawing with $\mathcal{O}(n^3/r^2)$ volume and aspect ratio $2r$.*

## 3. Track layouts.

In this section we describe a number of methods for producing and manipulating track layouts. The following result is implicit in the proof by Felsner, Liotta, and Wismath [42] that every outerplanar graph has an improper 3-track layout.

LEMMA 3.1 (see [42]). *Every tree $T$ has a 3-track layout.*

*Proof.* Root $T$ at an arbitrary node $r$. Let $\sigma$ be a lexicographical breadth-first vertex-ordering of $T$ starting at $r$, as described in section 2.4. For $i \in \{0, 1, 2\}$, let $V_i$ be the set of nodes of $T$ with depth $d \equiv i \pmod 3$ in $\sigma$. With each $V_i$ ordered by $\sigma$, we have a 3-track assignment of $T$. Clearly adjacent vertices are on distinct tracks. Since no two edges are nested in $\sigma$, there is no X-crossing (see Figure 3.1). $\square$

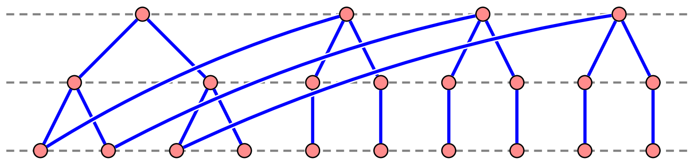


FIG. 3.1. *A* 3*-track layout of a tree.*

LEMMA 3.2. *Every graph $G$ with path-width* $\mathsf{pw}(G)$ *has track-number* $\mathsf{tn}(G) \leq \mathsf{pw}(G) + 1$.

*Proof.* Let $k = \mathsf{pw}(G)+1$. It is well known that $G$ is the subgraph of a $k$-colorable interval graph [10, 48]. That is, there is a set of intervals $\{[\ell(v), r(v)] \subseteq \mathbb{R} : v \in V(G)\}$ such that $[\ell(v), r(v)] \cap [\ell(w), r(w)] \neq \emptyset$ for every edge $vw$ of $G$. Let $\{V_i : 1 \leq i \leq k\}$ be a $k$-coloring of $G$. Consider each color class $V_i$ to be an ordered track $(v_1, v_2, \dots, v_p)$, where $\ell(v_1) < r(v_1) < \ell(v_2) < r(v_2) < \cdots < \ell(v_p) < r(v_p)$, as illustrated in Figure 3.2. Suppose there is an X-crossing between edges $vw$ and $xy$ with $v, x \in V_i$ and $w, y \in V_j$ for some pair of tracks $V_i$ and $V_j$. Without loss of generality, $r(v) < \ell(x)$ and $r(y) < \ell(w)$. Since $vw$ is an edge, $\ell(w) \leq r(v)$. Thus $r(y) < \ell(w) \leq r(v) < \ell(x)$, which implies that $xy$ is not an edge of $G$. This contradiction proves that there is no X-crossing, and $G$ has a $k$-track layout. $\square$

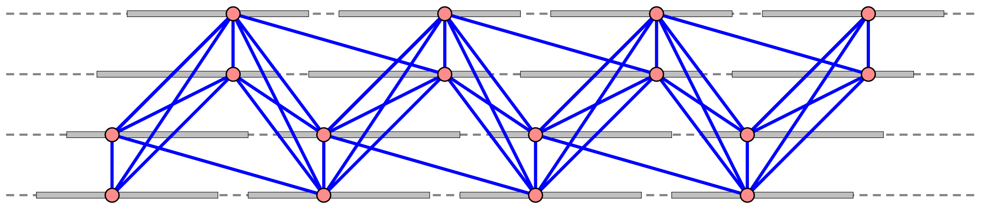


FIG. 3.2. *A* 4*-track layout of a* 4*-colorable interval graph.*

The next lemma uses a tree-partition to construct a track layout.

LEMMA 3.3. *Every graph $G$ with maximum degree $\Delta(G) \geq 1$, tree-width* $\mathsf{tw}(G)$*, and tree-partition-width* $\mathsf{tpw}(G)$*, has track-number* $\mathsf{tn}(G) \leq 3\,\mathsf{tpw}(G) \leq 72\,\Delta(G)\mathsf{tw}(G)$.

*Proof.* Let $(T, \{T_x : x \in V(T)\})$ be a tree-partition of $G$ with width $\mathsf{tpw}(G)$. By Lemma 3.1, $T$ has a 3-track layout. Replace each track by $\mathsf{tpw}(G)$ "subtracks," and for each node $x$ in $T$ place the vertices in bag $T_x$ on the subtracks replacing the track containing $x$, with at most one vertex in $T_x$ in a single track. For all nodes $x$ and $y$ of $T$, if $x < y$ in a single track of the 3-track layout of $T$, then for all vertices $v \in T_x$ and $w \in T_y$, $v < w$ whenever $v$ and $w$ are assigned to the same track. There is no X-crossing, since in the track layout of $T$, adjacent nodes are on distinct tracks and there is no X-crossing. Thus we have a track layout of $G$. The number of tracks is $3\,\mathsf{tpw}(G)$, which is at most $72\,\Delta(G)\mathsf{tw}(G)$ by the theorem of Ding and Oporowski [30] discussed in section 2.2. $\square$

In the remainder of this section, we prove two results that show how track layouts can be manipulated without introducing an X-crossing. The first is a generalization of the "wrapping" algorithm of Felsner, Liotta, and Wismath [42], who implicitly proved the case $s = 1$.

LEMMA 3.4. *If a graph $G$ has an (improper) track layout $\{(V_i, <_i) : 1 \leq i \leq t\}$ with maximum edge span $s$, then $G$ has an (improper) $(2s+1)$-track layout.*

*Proof.* Let $\ell = 2s + 1$. Construct an $\ell$-track assignment of $G$ by merging the tracks $\{V_i : i \equiv j \pmod{t}\}$ for each $j$, $0 \leq j \leq t-1$, with vertices in $V_\alpha$ appearing before vertices in $V_\beta$ in the new track $j$ for all $\alpha, \beta \equiv j \pmod{t}$ with $\alpha < \beta$. The given order of each $V_i$ is preserved in the new tracks. It remains to prove that there is no X-crossing. Consider two edges $vw$ and $xy$. Let $i_1$ and $i_2$, $1 \leq i_1 < i_2 \leq t$, be the minimum and maximum tracks containing $v$, $w$, $x$, or $y$ in the given $t$-track layout of $G$.

First consider the case that $i_2 - i_1 > 2s$. Then without loss of generality $v$ is in track $i_2$ and $y$ is in track $i_1$. Thus $w$ is in a greater track than $x$, and even if $x$ (or $y$) appear on the same track as $v$ (or $w$) in the new $\ell$-track assignment, $x$ (or $y$) will be to the left of $v$ (or $w$). Thus these edges do not form an X-crossing in the $\ell$-track assignment. Otherwise $i_2 - i_1 \leq 2s$. Thus any two of $v$, $w$, $x$, or $y$ will appear on the same track in the $\ell$-track assignment if and only if they are on the same track in the given $t$-track layout (since $\ell > 2s$). Hence the only way for these four vertices to appear on exactly two tracks in the $\ell$-track assignment is if they were on exactly two layers in the given $t$-track layout, in which case, by assumption, $vw$ and $xy$ do not form an X-crossing. Therefore there is no X-crossing, and we have an $\ell$-track layout of $G$. ∎

The next result shows that the number of vertices in different tracks of a track layout can be balanced without introducing an X-crossing. The proof is based on an idea due to Pach, Thiele, and Tóth [73] for balancing the size of the color classes in a coloring.

LEMMA 3.5. *If a graph $G$ has an (improper) $t$-track layout, then for every $t' > 0$, $G$ has an (improper) $\lfloor t + t' \rfloor$-track layout with at most $\lceil \frac{n}{t'} \rceil$ vertices in each track.*

*Proof.* For each track with $q > \lceil \frac{n}{t'} \rceil$ vertices, replace it by $\lceil q / \lceil \frac{n}{t'} \rceil \rceil$ "subtracks" each with exactly $\lceil \frac{n}{t'} \rceil$ vertices except for at most one subtrack with $q \bmod \lceil \frac{n}{t'} \rceil$ vertices, such that the vertices in each subtrack are consecutive in the original track and the original order is maintained. There is no X-crossing between subtracks from the same original track as there is at most one edge between such subtracks. There is no X-crossing between subtracks from different original tracks as otherwise there would be an X-crossing in the original. There are at most $\lfloor t' \rfloor$ tracks with $\lceil \frac{n}{t'} \rceil$ vertices. Since there are at most $t$ tracks with less than $\lceil \frac{n}{t'} \rceil$ vertices, one for each of the original tracks, there is a total of at most $\lfloor t + t' \rfloor$ tracks. ∎

## 4. Three-dimensional drawings and track layouts.

In this section we prove Theorem 2.9, which states that three-dimensional drawings with small volume are closely related to track layouts with few tracks.

LEMMA 4.1. *If a graph $G$ has an $A \times B \times C$ drawing, then $G$ has an improper $AB$-track layout, and $G$ has a $2AB$-track layout.*

*Proof.* Let $V_{x,y}$ be the set of vertices of $G$ with an $X$-coordinate of $x$ and a $Y$-coordinate of $y$, where without loss of generality $1 \leq x \leq A$ and $1 \leq y \leq Y$. With each set $V_{x,y}$ ordered by the $Z$-coordinates of its elements, $\{V_{x,y} : 1 \leq x \leq A, 1 \leq y \leq Y\}$ is an improper $AB$-track assignment. There is no X-crossing, as otherwise there would be a crossing in the original drawing, and hence we have an improper $AB$-track layout. By Lemma 2.2, $G$ has a $2AB$-track layout. ∎

We now prove the converse of Lemma 4.1. The proof is inspired by the generalizations of the moment curve algorithm by Cohen et al. [20] and Pach, Thiele, and Tóth [73], described in section 2.6. Loosely speaking, Cohen et al. [20] allow three "free" dimensions, whereas Pach, Thiele, and Tóth [73] use the assignment of vertices to color classes to "fix" one dimension with two dimensions free. We use an assignment of vertices to tracks to fix two dimensions with one dimension free. The style of three-dimensional drawing produced by our algorithm, where tracks are drawn vertically, is illustrated in Figure 4.1.

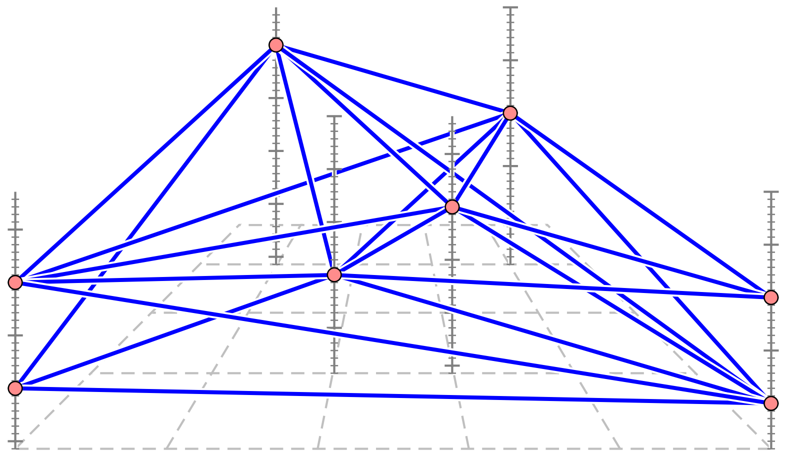


FIG. 4.1. *A three-dimensional drawing produced from a track layout.*

LEMMA 4.2. *If a graph $G$ has a (possibly) improper $k$-track layout, then $G$ has a $k \times 2k \times 2k \cdot n'$ three-dimensional drawing, where $n'$ is the maximum number of vertices in a track.*

*Proof.* Suppose that $\{(V_i, <_i) : 1 \leq i \leq k\}$ is the given improper $k$-track layout. Let $p$ be the smallest prime such that $p > k$. Then $p \leq 2k$ by Bertrand's postulate. For each $i$, $1 \leq i \leq k$, represent the vertices in $V_i$ by the grid-points

$$\{(i, i^2 \bmod p, t) : 1 \leq t \leq p \cdot |V_i|, t \equiv i^3 \pmod p\}$$

such that the $Z$-coordinates respect the given total order $<_i$. Draw each edge as a line-segment between its end-vertices. Suppose that two edges $e$ and $e'$ cross such that their end-vertices are at distinct points $(i_\alpha, i_\alpha^2 \bmod p, t_\alpha)$, $1 \leq \alpha \leq 4$. Then these points are coplanar, and if $M$ is the matrix

$$M = \begin{pmatrix} 1 & i_1 & i_1^2 \bmod p & t_1 \\ 1 & i_2 & i_2^2 \bmod p & t_2 \\ 1 & i_3 & i_3^2 \bmod p & t_3 \\ 1 & i_4 & i_4^2 \bmod p & t_4 \end{pmatrix},$$

then the determinant $\det(M) = 0$. We proceed by considering the number of distinct tracks $N = |\{i_1, i_2, i_3, i_4\}|$.

• $N = 1$: By the definition of an improper track layout, $e$ and $e'$ do not cross.

• $N = 2$: If either edge is intratrack, then $e$ and $e'$ do not cross. Otherwise neither edge is intratrack, and since there is no X-crossing, $e$ and $e'$ do not cross.

• $N = 3$: Without loss of generality $i_1 = i_2$. It follows that $\det(M) = (t_2 - t_1) \cdot \det(M')$, where

$$M' = \begin{pmatrix} 1 & i_2 & i_2^2 \bmod p \\ 1 & i_3 & i_3^2 \bmod p \\ 1 & i_4 & i_4^2 \bmod p \end{pmatrix}.$$

Since $t_1 \neq t_2$, $\det(M') = 0$. However, $M'$ is a Vandermonde matrix modulo $p$, and thus

$$\det(M') \equiv (i_2 - i_3)(i_2 - i_4)(i_3 - i_4) \pmod{p},$$

which is nonzero since $i_2$, $i_3$, and $i_4$ are distinct and $p$ is a prime, a contradiction.

• $N = 4$: Let $M'$ be the matrix obtained from $M$ by taking each entry modulo $p$. Then $\det(M') = 0$. Since $t_\alpha \equiv i_\alpha^3 \pmod{p}$, $1 \leq \alpha \leq 4$,

$$M' \equiv \begin{pmatrix} 1 & i_1 & i_1^2 & i_1^3 \\ 1 & i_2 & i_2^2 & i_2^3 \\ 1 & i_3 & i_3^2 & i_3^3 \\ 1 & i_4 & i_4^2 & i_4^3 \end{pmatrix} \pmod{p}.$$

Since each $i_\alpha < p$, $M'$ is a Vandermonde matrix modulo $p$, and thus

$$\det(M') \equiv (i_1 - i_2)(i_1 - i_3)(i_1 - i_4)(i_2 - i_3)(i_2 - i_4)(i_3 - i_4) \pmod{p},$$

which is nonzero since $i_\alpha \neq i_\beta$ and $p$ is a prime. This contradiction proves there are no edge crossings. The produced drawing is at most $k \times 2k \times 2k \cdot n'$. $\square$

*Proof of Theorem* 2.9. Let $\mathcal{F}(n)$ be a family of functions closed under multiplication. Let $G$ be an $n$-vertex graph with a $t$-track layout, where $t \in \mathcal{F}(n)$. By Lemma 3.5 with $t' = t$, $G$ has a $2t$-track layout with at most $\lceil \frac{n}{t} \rceil$ vertices in each track. By Lemma 4.2, $G$ has a $2t \times 4t \times 4t \cdot \lceil \frac{n}{t} \rceil$ drawing, which is $\mathcal{O}(t) \times \mathcal{O}(t) \times \mathcal{O}(n)$. Conversely, suppose that an $n$-vertex graph $G$ has an $A \times B \times \mathcal{O}(n)$ drawing, where $A, B \in \mathcal{F}(n)$. By Lemma 4.1, $G$ has a track layout with $2AB \in \mathcal{F}(n)$ tracks. $\square$

*Proof of Theorem* 2.14. Let $t = \mathsf{tn}(G)$, and suppose $1 \leq r \leq n/t$. By Lemma 3.5 with $t' = \frac{n}{r}$, $G$ has a $\lfloor \frac{n}{r} + t \rfloor$-track layout with at most $r$ vertices in each track. By assumption $t \leq \frac{n}{r}$, and the number of tracks is at most $\frac{2n}{r}$. By Lemma 4.2, $G$ has a $\frac{2n}{r} \times \frac{4n}{r} \times 4n$ three-dimensional drawing, which has volume $32n^3/r^2$ and aspect ratio $2r$. $\square$

## 5. Queue layouts and track layouts.

In this section we prove Theorem 2.6, which states that track and queue layouts are closely related. Our first lemma highlights this fact—its proof follows immediately from the definitions (see Figure 5.1).

LEMMA 5.1. *A bipartite graph $G = (A, B; E)$ has a 2-track layout with tracks $A$ and $B$ if and only if $G$ has a 1-queue layout such that in the corresponding vertex-ordering, the vertices in $A$ appear before the vertices in $B$.* $\square$

We now show that a queue layout can be obtained from a track layout. This result can be viewed as a generalization of the construction of a 2-queue layout of an outerplanar graph by Heath, Leighton, and Rosenberg [54] and Rengarajan and Veni Madhavan [78] (with $s = 1$).

LEMMA 5.2. *If a graph $G$ has a (possibly) improper $t$-track layout $\{(V_i, <_i) : 1 \leq i \leq t\}$ with maximum edge span $s$ $(\leq t-1)$, then $\mathsf{qn}(G) \leq s+1$, and if the given track layout is not improper, then $\mathsf{qn}(G) \leq s$.*

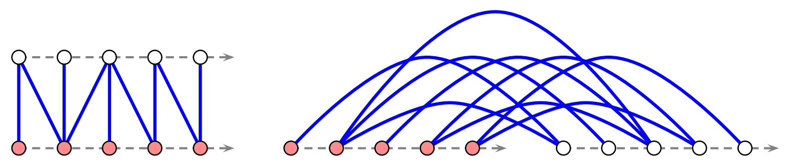


FIG. 5.1. *A 2-track layout and a 1-queue layout of a bipartite graph.*

*Proof.* First suppose that there are no intratrack edges. Let $\sigma$ be the vertex ordering $(V_1, V_2, \dots, V_t)$ of $G$. Let $E_\alpha$ be the set of edges with span $\alpha$ in the given track layout. As in Lemma 5.1, two edges from the same pair of tracks are nested in $\sigma$ if and only if they form an X-crossing in the track layout. Since no two edges form an X-crossing in the track layout, no two edges that are between the same pair of tracks are nested in $\sigma$. If two edges not from the same pair of tracks have the same span, then they are not nested in $\sigma$. (This idea is due to Heath and Rosenberg [58].) Thus no two edges are nested in each $E_\alpha$, and we have an $s$-queue layout of $G$. If there are intra-track edges, then they all form one additional queue in $\sigma$. $\square$

We now set out to prove the converse of Lemma 5.2. It is well known that the subgraph induced by any two tracks of a track layout is a forest of caterpillars [52]. A coloring of a graph is *acyclic* if every bichromatic subgraph is a forest; that is, every cycle receives at least three distinct colors. Thus a $t$-track layout of a graph $G$ defines an acyclic $t$-coloring of $G$. The minimum number of colors in an acyclic coloring of $G$ is the *acyclic chromatic number* of $G$, denoted by $\chi_{\mathrm{a}}(G)$. Thus,

$$\chi_{\mathrm{a}}(G) \leq \mathsf{tn}(G).$$

Acyclic colorings were introduced by Grünbaum [49], who proved that every planar graph is acyclically 9-colorable. This result was steadily improved [1, 65, 67] until Borodin [12] proved that every planar graph is acyclically 5-colorable, which is the best possible bound. Many other graph families have bounded acyclic chromatic number, including graphs embeddable on a fixed surface [2, 3, 6], 1-planar graphs [13], graphs with bounded maximum degree [5], and graphs with bounded tree-width. A folklore result states that $\chi_{\mathrm{a}}(G) \leq \mathsf{tw}(G) + 1$ (see [43]). More generally, Nešetřil and Ossona de Mendez [69] proved that every proper minor-closed graph family has bounded acyclic chromatic number. In fact, they proved that every graph $G$ has a *star* $k$-coloring (every bichromatic subgraph is a forest of stars), where $k$ is a (small) quadratic function of the maximum chromatic number of a minor of $G$.

LEMMA 5.3. *Every graph $G$ with acyclic chromatic number $\chi_{\mathrm{a}}(G) \leq c$ and queue-number $\mathsf{qn}(G) \leq q$ has track-number $\mathsf{tn}(G) \leq c\,(2q)^{c-1}$.*

*Proof.* Let $\{V_i : 1 \leq i \leq c\}$ be an acyclic coloring of $G$. Let $\sigma$ be the vertex-ordering in a $q$-queue layout of $G$. Consider an edge $vw$ with $v \in V_i$, $w \in V_j$, and $i < j$. If $v <_\sigma w$, then $vw$ is *forward*, and if $w <_\sigma v$, then $vw$ is *backward*. Consider the edges to be colored with $2q$ colors, where each color class consists of the forward edges in a single queue, or the backward edges in a single queue.

Alon and Marshall [4] proved that given a (not necessarily proper) edge $k$-coloring of a graph $G$, any acyclic $c$-coloring of $G$ can be refined to a $ck^{c-1}$-coloring so that the edges between any pair of (vertex) color classes are monochromatic, and each (vertex) color class is contained in some original color class. (Nešetřil and Raspaud [70] generalized this result for colored mixed graphs.) Apply this result with the given acyclic $c$-coloring of $G$ and the edge $2q$-coloring discussed above. Consider the resulting $c(2q)^{c-1}$ color classes to be tracks ordered by $\sigma$. The edges between any two tracks are from a single queue, and are all forward or all backward.

Suppose that there are edges $vw$ and $xy$ that form an X-crossing. Since each track is a subset of some $V_i$, we can assume that $v, x \in V_i$, $w, y \in V_j$, and $i < j$. Suppose that $vw$ and $xy$ are both forward. The case in which $vw$ and $xy$ are both backward is symmetric. Thus $v <_\sigma w$ and $x <_\sigma y$. Since $vw$ and $xy$ form an X-crossing, and the tracks are ordered by $\sigma$, we have $v <_\sigma x$ and $y <_\sigma w$. Hence $v <_\sigma x <_\sigma y <_\sigma w$. That is, $vw$ and $xy$ are nested. This is the desired contradiction, since edges between any pair of tracks are from a single queue. Thus we have a $c(2q)^{c-1}$-track layout of $G$. $\square$

*Proof of Theorem* 2.6. Let $\mathcal{F}(n)$ be a family of functions closed under multiplication. Let $G$ be an $n$-vertex graph from a proper minor-closed graph family $\mathcal{G}$. First, suppose that $G$ has a $t$-track layout, where $t \in \mathcal{F}(n)$. By Lemma 5.2, $G$ has queue-number $\mathsf{qn}(G) \leq t - 1 \in \mathcal{F}(n)$. Conversely, suppose $G$ has queue-number $\mathsf{qn}(G) = q \in \mathcal{F}(n)$. By the above-mentioned result of Nešetřil and Ossona de Mendez [69], $G$ has bounded acyclic chromatic number $\chi_{\mathrm{a}}(G) \leq c \in \mathcal{O}(1)$. By Lemma 5.3, $G$ has a $t$-track layout, where $t \leq c(2q)^{c-1} \in \mathcal{F}(n)$. $\square$

## 6. Tree-partitions of *k*-trees.

In this section we prove our theorem mentioned in section 2.2 regarding tree-partitions of $k$-trees. This result forms the cornerstone of the proof of Theorem 7.3.

THEOREM 6.1. *Let $G$ be a $k$-tree with maximum degree $\Delta$. Then $G$ has a rooted tree-partition $(T, \{T_x : x \in V(T)\})$ such that for all nodes $x$ of $T$,*

(a) *if $x$ is a nonroot node of $T$ and $y$ is the parent node of $x$, then the set of vertices in $T_y$ with a neighbor in $T_x$ forms a clique $C_x$ of $G$, and*

(b) *the induced subgraph $G[T_x]$ is a connected $(k-1)$-tree.*

*Furthermore the width of $(T, \{T_x : x \in V(T)\})$ is at most* $\max\{1, k(\Delta - 1)\}$.

*Proof.* We assume that $G$ is connected, since if $G$ is not connected, then a tree-partition of $G$ that satisfies the theorem can be determined by adding a new root node with an empty bag, adjacent to the root node of a tree-partition of each connected component of $G$.

It is well known that $G$ is a connected $k$-tree if and only if $G$ has a vertex-ordering $\sigma = (v_1, v_2, \ldots, v_n)$, such that for all $i \in \{1, 2, \ldots, n\}$,

(i) if $G^i$ is the induced subgraph $G[\{v_1, v_2, \ldots, v_i\}]$, then $G^i$ is connected and the vertex-ordering of $G^i$ induced by $\sigma$ is a breadth-first vertex-ordering of $G^i$, and

(ii) the neighbors of $v_i$ in $G^i$ form a clique $C_i = \{v_j : v_i v_j \in E(G), j < i\}$ with $1 \leq |C_i| \leq k$ (unless $i = 1$, in which case $C_i = \emptyset$).

In the language of chordal graphs, $\sigma$ is a (reverse) "perfect elimination" vertex-ordering and can be determined, for example, by the Lex-BFS algorithm by Rose, Tarjan, and Leuker [80] (also see [48]). Moreover, we can choose $v_1$ to be any vertex in $G$.

Let $r$ be a vertex of minimum degree[6] in $G$. Then $\deg(r) \leq k$. Let $\sigma = (v_1, v_2, \ldots, v_n)$ be a vertex-ordering of $G$ with $v_1 = r$ and satisfying (i) and (ii). By (i), the depth of each vertex $v_i$ in $\sigma$ is the same as the depth of $v_i$ in the vertex-ordering of $G^j$ induced by $\sigma$ for all $j \geq i$. We therefore simply speak of *the* depth of $v_i$. Let $V_d$ be the set of vertices of $G$ at depth $d$.

[6]We choose $r$ to have minimum degree to obtain a slightly improved bound on the width of the tree-partition. If we choose $r$ to be an arbitrary vertex, then the width is at most $\max\{1, \Delta, k(\Delta-1)\}$, and the remainder of Theorem 6.1 holds.

CLAIM 1. *For all $d \geq 1$, and for every connected component $Z$ of $G[V_d]$, the set of vertices at depth $d-1$ with a neighbor in $Z$ form a clique of $G$.*

*Proof.* The claim is trivial for $d = 1$ or $d = 2$. Now suppose that $d \geq 3$. Assume for the sake of contradiction that there are two nonadjacent vertices $x$ and $y$ at depth $d-1$ such that $x$ has a neighbor in $Z$ and $y$ has a neighbor in $Z$. Let $P_1$ be a shortest path between $x$ and $y$ with its interior vertices in $Z$. Let $P_2$ be a shortest path between $x$ and $y$ with its interior vertices at depth at most $d-2$. Since the interior vertices of $P_1$ are at depth $d$, there is no edge between an interior vertex of $P_1$ and an interior vertex of $P_2$. Thus $P_1 \cup P_2$ is a chordless cycle of length at least four, contradicting the fact that $G$ is chordal (by Lemma 2.1). $\square$

Define a graph $T$ and a partition $\{T_x : x \in V(T)\}$ of $V(G)$ indexed by the nodes of $T$ as follows. There is one node $x$ in $T$ for every connected component of each $G[V_d]$, whose bag $T_x$ is the vertex-set of the corresponding connected component. We say $x$ and $T_x$ are at *depth* $d$. Clearly a vertex in a depth-$d$ bag is also at depth $d$. The (unique) node of $T$ at depth zero is called the *root* node. Let two nodes $x$ and $y$ of $T$ be connected by an edge if there is an edge $vw$ of $G$ with $v \in T_x$ and $w \in T_y$. Thus $(T, \{T_x : x \in V(T)\})$ is a "graph-partition."

We now prove that in fact $T$ is a tree. First observe that $T$ is connected since $G$ is connected. By definition, nodes of $T$ at the same depth $d$ are not adjacent. Moreover, nodes of $T$ can be adjacent only if their depths differ by one. Thus $T$ has a cycle only if there is a node $x$ in $T$ at some depth $d$ such that $x$ has at least two distinct neighbors in $T$ at depth $d-1$. However this is impossible since, by Claim 1, the set of vertices at depth $d-1$ with a neighbor in $T_x$ form a clique (which we call $C_x$) and are hence in a single bag at depth $d-1$. Thus $T$ is a tree, and $(T, \{T_x : x \in V(T)\})$ is a tree-partition of $G$ (see Figure 6.1).

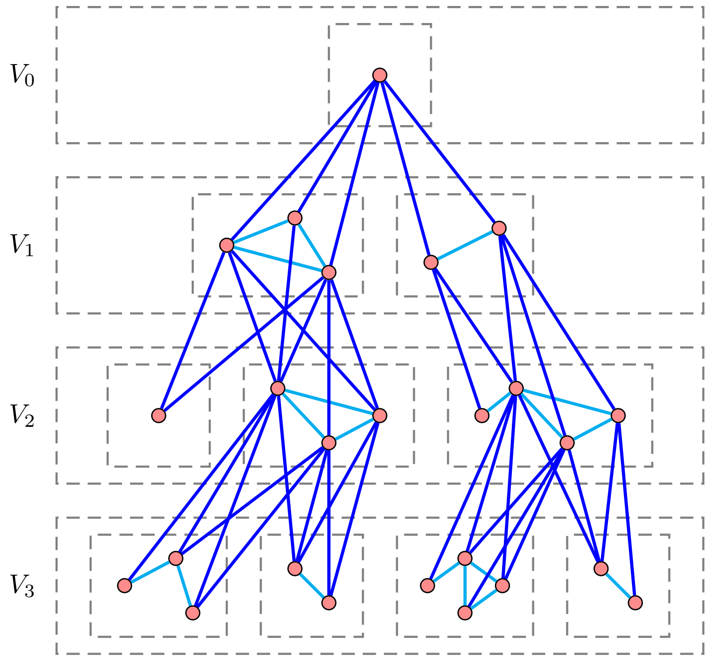


FIG. 6.1. *Illustration for Theorem* 6.1 *in the case of* $k = 3$.

We now prove that each bag $T_x$ induces a connected $(k-1)$-tree. This is true for the root node which only has one vertex. Suppose $x$ is a nonroot node of $T$ at depth $d$. Each vertex in $T_x$ has at least one neighbor at depth $d-1$. Thus in the vertex-ordering of $T_x$ induced by $\sigma$, each vertex $v_i \in T_x$ has at most $k-1$ neighbors $v_j \in T_x$ with $j < i$. Thus the vertex-ordering of $T_x$ induced by $\sigma$ satisfies (i) and (ii) for $k-1$, and $G[T_x]$ is $(k-1)$-tree. By definition, each $G[T_x]$ is connected.

Finally, consider the cardinality of a bag in $T$. We claim that each bag contains at most $\max\{1, k(\Delta-1)\}$ vertices. The root bag has one vertex. Let $x$ be a nonroot node of $T$ with parent node $y$. Suppose that $y$ is the root node. Then $T_y = \{r\}$, and thus $|T_x| \leq \deg(r) \leq k \leq k(\Delta-1)$, assuming $\Delta \geq 2$. If $\Delta \leq 1$, then all bags have one vertex. Now assume that $y$ is a nonroot node. The set of vertices in $T_y$ with a neighbor in $T_x$ forms the clique $C_x$. Let $k' = |C_x|$. Thus $k' \geq 1$, and since $C_x \subseteq T_y$ and $G[T_y]$ is a $(k-1)$-tree, $k' \leq k$. A vertex $v \in C_x$ has $k'-1$ neighbors in $C_x$ and at least one neighbor in the parent bag of $y$. Thus $v$ has at most $\Delta - k'$ neighbors in $T_x$. Hence the number of edges between $C_x$ and $T_x$ is at most $k'(\Delta-k')$. Every vertex in $T_x$ is adjacent to a vertex in $C_x$. Thus $|T_x| \leq k'(\Delta-k') \leq k(\Delta-1)$. This completes the proof. ∎

**7. Tree-width and track layouts.** In this section we prove that track-number is bounded by tree-width. Let $\{(V_i, <_i) : i \in I\}$ be a track layout of a graph $G$. We say a clique $C$ of $G$ *covers* the set of tracks $\{i \in I : C \cap V_i \neq \emptyset\}$. Let $S$ be a set of cliques of $G$. Suppose that there exists a total order $\preceq$ on $S$ such that for all cliques $C_1, C_2 \in S$, if there exists a track $i \in I$, and vertices $v \in V_i \cap C_1$ and $w \in V_i \cap C_2$ with $v <_i w$, then $C_1 \prec C_2$. In this case, we say $\preceq$ is *nice*, and $S$ is *nicely ordered* by the track layout.

LEMMA 7.1. *Let $L \subseteq I$ be a set of tracks in a track layout $\{(V_i, <_i) : i \in I\}$ of a graph $G$. If $S$ is a set of cliques each of which covers $L$, then $S$ is nicely ordered by the given track layout.*

*Proof.* Define a relation $\preceq$ on $S$ as follows. For every pair of cliques $C_1, C_2 \in S$, define $C_1 \preceq C_2$ if $C_1 = C_2$ or there exists a track $i \in L$ and vertices $v \in C_1$ and $w \in C_2$ with $v <_i w$. Clearly all cliques in $S$ are comparable.

Suppose that $\preceq$ is not antisymmetric; that is, there exist distinct cliques $C_1, C_2 \in S$, distinct tracks $i, j \in L$, and distinct vertices $v_1, w_1 \in C_1$ and $v_2, w_2 \in C_2$ such that $v_1 <_i v_2$ and $w_2 <_j w_1$. Since $C_1$ and $C_2$ are cliques, the edges $v_1w_1$ and $v_2w_2$ form an X-crossing, which is a contradiction. Thus $\preceq$ is antisymmetric.

We claim that $\preceq$ is transitive. Suppose that there exist cliques $C_1, C_2, C_3 \in S$ such that $C_1 \preceq C_2$ and $C_2 \preceq C_3$. We can assume that $C_1$, $C_2$, and $C_3$ are pairwise distinct. Thus there are vertices $u_1 \in C_1$, $u_2 \in C_2$, $v_2 \in C_2$, and $v_3 \in C_3$ such that $u_1 <_i u_2$ and $v_2 <_j v_3$ for some pair of (not necessarily distinct) tracks $i, j \in L$. Since $C_3$ has a vertex in $V_i$ and since $C_3 \not\preceq C_2$, there is a vertex $u_3 \in C_3$ with $u_2 \leq_i u_3$. Thus $u_1 <_i u_3$, which implies that $C_1 \preceq C_3$. Thus $\preceq$ is transitive.

Hence $\preceq$ is a total order on $S$, which by definition is nice. ∎

Consider the problem of partitioning the cliques of a graph into sets such that each set is nicely ordered by a given track layout. The following immediate corollary of Lemma 7.1 says that there exists such a partition where the number of sets does not depend upon the size of the graph.

COROLLARY 7.2. *Let $G$ be a graph with maximum clique size $k$. Given a $t$-track layout of $G$, there is a partition of the cliques of $G$ into $\sum_{i=1}^{k} \binom{t}{i}$ sets, each of which is nicely ordered by the given track layout.* ∎

We do not actually use Corollary 7.2 in the following result, but the idea of partitioning the cliques into nicely ordered sets is central to its proof.

THEOREM 7.3. *For every integer $k \geq 0$, there is a constant $t_k = 3^k \cdot 6^{(4^k-3k-1)/9}$ such that every graph $G$ with tree-width $\mathsf{tw}(G) \leq k$ has a $t_k$-track layout.*

*Proof.* If the input graph $G$ is not a $k$-tree, then add edges to $G$ to obtain a $k$-tree containing $G$ as a subgraph. It is well known that a graph with tree-width at most $k$ is a *spanning* subgraph of a $k$-tree. These extra edges can be deleted once we are done. We proceed by induction on $k$ with the following hypothesis:

*For all $k \in \mathbb{N}$, there exists a constant $s_k$ and sets $\mathcal{I}_k$ and $\mathcal{S}_k$ such that*

1. *$|\mathcal{I}_k| = t_k$ and $|\mathcal{S}_k| = s_k$,*
2. *each element of $\mathcal{S}_k$ is a subset of $\mathcal{I}_k$, and*
3. *every $k$-tree $G$ has a $t_k$-track layout indexed by $\mathcal{I}_k$, such that for every clique $C$ of $G$, the set of tracks that $C$ covers is in $\mathcal{S}_k$.*

Consider the base case with $k = 0$. A 0-tree $G$ has no edges and thus has a 1-track layout. Let $\mathcal{I}_0 = \{1\}$, and order $V_1 = V(G)$ arbitrarily. Thus $t_0 = 1$, $s_0 = 1$, and $\mathcal{S}_0 = \{\{1\}\}$ satisfy the hypothesis for every 0-tree. Now suppose that the result holds for $k-1$, and $G$ is a $k$-tree.

Let $(T, \{T_x : x \in V(T)\})$ be a tree-partition of $G$ described in Theorem 6.1, where $T$ is rooted at $r$. Each induced subgraph $G[T_x]$ is a $(k-1)$-tree. Thus, by induction, there are sets $\mathcal{I}_{k-1}$ and $\mathcal{S}_{k-1}$ with $|\mathcal{I}_{k-1}| = t_{k-1}$ and $|\mathcal{S}_{k-1}| = s_{k-1}$ such that for every node $x$ of $T$ the induced subgraph $G[T_x]$ has a $t_{k-1}$-track layout indexed by $\mathcal{I}_{k-1}$. For every clique $C$ of $G[T_x]$, if $C$ covers $L \subseteq \mathcal{I}_{k-1}$, then $L \in \mathcal{S}_{k-1}$. Assume $\mathcal{I}_{k-1} = \{1, 2, \ldots, t_{k-1}\}$ and $\mathcal{S}_{k-1} = \{X_1, X_2, \ldots, X_{s_{k-1}}\}$. By Theorem 6.1, for each nonroot node $x$ of $T$, if $p$ is the parent node of $x$, then the set of vertices in $T_p$ with a neighbor in $T_x$ form a clique $C_x$. Let $\alpha(x) = i$, where $C_x$ covers $X_i$. For the root node $r$ of $T$, let $\alpha(r) = 1$.

**Track layout of $T$.** To construct a track layout of $G$ we first construct a track layout of the tree $T$ indexed by the set $\{(d, i) : d \geq 0, 1 \leq i \leq s_{k-1}\}$, where the track $L_{d,i}$ consists of nodes $x$ of $T$ at depth $d$ with $\alpha(x) = i$. Here the *depth* of a node $x$ is the distance in $T$ from the root node $r$ to $x$. We order the nodes of $T$ within the tracks by increasing depth. There is only one node at depth $d = 0$. Suppose that we have determined the orders of the nodes up to depth $d-1$ for some $d \geq 1$.

Let $i \in \{1, 2, \ldots, s_{k-1}\}$. The nodes in $L_{d,i}$ are ordered primarily with respect to the relative positions of their parent nodes (at depth $d-1$). More precisely, let $\rho(x)$ denote the parent node of each node $x \in L_{d,i}$. For all nodes $x$ and $y$ in $L_{d,i}$, if $\rho(x)$ and $\rho(y)$ are in the same track and $\rho(x) < \rho(y)$ in that track, then $x < y$ in $L_{d,i}$. For $x$ and $y$ with $\rho(x)$ and $\rho(y)$ on distinct tracks, the relative order of $x$ and $y$ is not important. It remains to specify the order of nodes in $L_{d,i}$ with a common parent.

Suppose that $P$ is a set of nodes in $L_{d,i}$ with a common parent node $p$. By construction, for every node $x \in P$, the parent clique $C_x$ covers $X_i$ in the track layout of $G[T_p]$. By Lemma 7.1 the cliques $\{C_x : x \in P\}$ are nicely ordered by the track layout of $G[T_p]$. Let the order of $P$ in track $L_{d,i}$ be specified by a nice ordering of $\{C_x : x \in P\}$, as illustrated in Figure 7.1.

This construction defines a partial order on the nodes in track $L_{d,i}$, which can be arbitrarily extended to a total order. Hence we have a track assignment of $T$. Since the nodes in each track are ordered primarily with respect to the relative positions of their parent nodes in the previous tracks, there is no X-crossing, and hence we have a track layout of $T$.

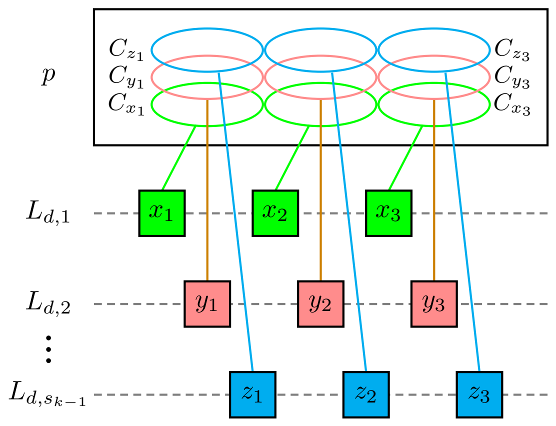


FIG. 7.1. *Track layout of nodes with a common parent $p$.*

**Track layout of $G$.** To construct a track assignment of $G$ from the track layout of $T$, replace each track $L_{d,i}$ by $t_{k-1}$ subtracks, and for each node $x$ of $T$ insert the track layout of $G[T_x]$ in place of $x$ on the subtracks corresponding to the track containing $x$ in the track layout of $T$. More formally, the track layout of $G$ is indexed by the set

$$\{(d, i, j) : d \geq 0, 1 \leq i \leq s_{k-1}, 1 \leq j \leq t_{k-1}\}.$$

Each track $V_{d,i,j}$ consists of those vertices $v$ of $G$ such that, if $T_x$ is the bag containing $v$, then $x$ is at depth $d$ in $T$, $\alpha(x) = i$, and $v$ is in track $j$ in the track layout of $G[T_x]$. If $x$ and $y$ are distinct nodes of $T$ with $x < y$ in $L_{d,i}$, then $v < w$ in $V_{d,i,j}$ for all vertices $v \in T_x$ and $w \in T_y$ in track $j$. If $v$ and $w$ are vertices of $G$ in track $j$ in bag $T_x$ at depth $d$, then the relative order of $v$ and $w$ in $V_{d,\alpha(x),j}$ is the same as in the track layout of $G[T_x]$.

Clearly adjacent vertices of $G$ are in distinct tracks. Thus we have defined a track assignment of $G$. We claim there is no X-crossing. Clearly an intrabag edge of $G$ is not in an X-crossing with an edge not in the same bag. By induction, there is no X-crossing between intrabag edges in a common bag. Since there is no X-crossing in the track layout of $T$, interbag edges of $G$ which are mapped to edges of $T$ without a common parent node are not involved in an X-crossing.

Consider a parent node $p$ in $T$. For each child node $x$ of $p$, the set of vertices in $T_p$ adjacent to a vertex in $T_x$ forms the clique $C_x$. Thus there is no X-crossing between a pair of edges both from $C_x$ to $T_x$, since the vertices of $C_x$ are on distinct tracks. Consider two child nodes $x$ and $y$ of $p$. For there to be an X-crossing between an edge from $T_p$ to $T_x$ and an edge from $T_p$ to $T_y$, the nodes $x$ and $y$ must be on the same track in the track layout of $T$. Suppose $x < y$ in this track. By construction, $C_x$ and $C_y$ cover the same set of tracks, and $C_x \preceq C_y$ in the corresponding nice ordering. Thus for any track containing vertices $v \in C_x$ and $w \in C_y$, $v \leq w$ in that track. Since all the vertices in $T_x$ are to the left of the vertices in $T_y$ (in a common track), there is no X-crossing between an edge from $T_p$ to $T_x$ and an edge from $T_p$ to $T_y$. Therefore there is no X-crossing, and hence we have a track layout of $G$.

**Wrapped track layout of $G$.** 0As illustrated in Figure 7.2, we now "wrap" the track layout of $G$ in the spirit of Lemma 3.1. In particular, define a track assignment

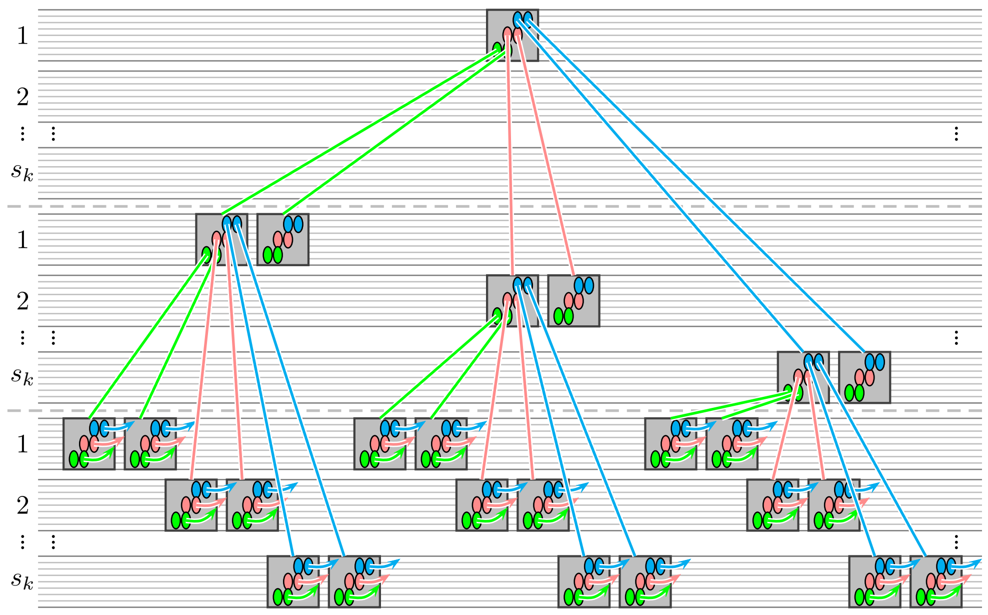


FIG. 7.2. *Wrapped track layout in Theorem* 7.3.

of $G$ indexed by

$$\{(d',i,j) : d' \in \{0,1,2\}, 1 \leq i \leq s_{k-1}, 1 \leq j \leq t_{k-1}\},$$

where each track

$$W_{d',i,j} = \bigcup\{V_{d,i,j} : d \equiv d' \pmod 3\}.$$

If $v \in V_{d,i,j}$ and $w \in V_{d+3,i,j}$, then $v < w$ in the order of $W_{d',i,j}$ (where $d' = d \bmod 3$). The order of each $V_{d,i,j}$ is preserved in $W_{d',i,j}$. The set of tracks $\{W_{d',i,j} : d' \in \{0,1,2\}, 1 \leq i \leq s_{k-1}, 1 \leq j \leq t_{k-1}\}$ forms a track assignment of $G$.

For every edge $vw$ of $G$, the depths of the bags in $T$ containing $v$ and $w$ differ by at most one. Thus in the wrapped track assignment of $G$, adjacent vertices remain on distinct tracks, and there is no X-crossing. The number of tracks is $3 \cdot s_{k-1} \cdot t_{k-1}$.

Every clique $C$ of $G$ is either contained in a single bag of the tree-partition or is contained in two adjacent bags. Let

$$\mathcal{S}' = \{\{(d',i,h) : h \in X_j\} : d' \in \{0,1,2\}, 1 \leq i,j \leq s_{k-1}\}.$$

For every clique $C$ of $G$ contained in a single bag, the set of tracks containing $C$ is in $\mathcal{S}'$. Let

$$\mathcal{S}'' = \{\{(d',i,\ell) : \ell \in X_j\} \cup \{((d'+1) \bmod 3, p, h) : h \in X_q\} : \\ d' \in \{0,1,2\}, 1 \leq i,j,p,q \leq s_{k-1}\}.$$

For every clique $C$ of $G$ contained in two bags, the set of tracks containing $C$ is in $\mathcal{S}''$. Observe that $\mathcal{S}' \cup \mathcal{S}''$ is independent of $G$. Hence $\mathcal{S}_k = \mathcal{S}' \cup \mathcal{S}''$ satisfies the hypothesis for $k$.

Now $|\mathcal{S}'| = 3s_{k-1}^2$ and $|\mathcal{S}''| = 3s_{k-1}^4$, and thus $|\mathcal{S}' \cup \mathcal{S}''| = 3s_{k-1}^2(s_{k-1}^2 + 1)$. Therefore any solution to the following set of recurrences satisfies the theorem:

$$s_0 \ \geq\ 1, \quad t_0 \ \geq\ 1, \quad s_k \ \geq\ 3s_{k-1}^2(s_{k-1}^2+1), \quad t_k \ \geq\ 3s_{k-1}\cdot t_{k-1}. \tag{7.1}$$

We claim that

$$s_k = 6^{(4^k-1)/3} \quad \text{and} \quad t_k = 3^k \cdot 6^{(4^k-3k-1)/9}$$

is a solution to (7.1). Observe that $s_0 = 1$ and $t_0 = 1$. Now

$$3s_{k-1}^2(s_{k-1}^2+1) \ \leq\ 6s_{k-1}^4$$

and

$$6(6^{(4^{k-1}-1)/3})^4 \ =\ 6^{1+4(4^{k-1}-1)/3} \ =\ 6^{(4^k-1)/3} \ =\ s_k.$$

Thus the recurrence for $s_k$ is satisfied. Now

$$\begin{aligned} 3\cdot s_{k-1}\cdot t_{k-1} &= 3\cdot 6^{(4^{k-1}-1)/3}\cdot 3^{k-1}\cdot 6^{(4^{k-1}-3(k-1)-1)/9}\\ &= 3^k\cdot 6^{(3\cdot 4^{k-1}-3+4^{k-1}-3k+3-1)/9}\\ &= 3^k\cdot 6^{(4^k-3k-1)/9}\\ &= t_k. \end{aligned}$$

Thus the recurrence for $t_k$ is satisfied. This completes the proof. □

In the proof of Theorem 7.3 we have made little effort to reduce the bound on $t_k$, beyond that it is a doubly exponential function of $k$. In [35] we describe a number of refinements that result in improved bounds on $t_k$. One such refinement uses strict $k$-trees. From an algorithmic point of view, the disadvantage of using strict $k$-trees is that at each recursive step, extra edges must be added to enlarge the graph from a partial strict $k$-tree into a strict $k$-tree, whereas when using (nonstrict) $k$-trees, extra edges need be added only at the beginning of the algorithm.

For small values of $k$, much-improved results can be obtained. For example, we prove that every series-parallel graph (that is, with tree-width at most two) has an 18-track layout [35], whereas $t_2 = 54$. This bound has recently been improved to 15 by Di Giacomo, Liotta, and Meijer [25]. Their method is based on Theorems 6.1 and 7.3, and in the general case still gives a doubly exponential upper bound on the track-number of graphs with tree-width $k$. For other particular classes of graphs, Di Giacomo [24] and Di Giacomo and Meijer [27] recently improved the constants in our results.

Our doubly exponential upper bound is probably not best possible. Di Giacomo, Liotta, and Meijer [25] constructed graphs with tree-width $k$ and track-number at least $2k+1$. The following construction establishes a quadratic lower bound. It is similar to a graph due to Albertson et al. [3], which gives a tight lower bound on the star chromatic number of graphs with tree-width $k$.

THEOREM 7.4. *For all $k \geq 0$, there is a graph $G_k$ with tree-width at most $k$ and track-number* $\mathsf{tn}(G_k) = \frac{1}{2}(k+1)(k+2)$.

*Proof.* Let $G_0 = K_1$. Obviously $G_0$ has tree-width 0. Construct $G_k$ from $G_{k-1}$ as follows. Start with a $k$-clique $\{v_1, v_2, \dots, v_k\}$. Let $n = 2(\frac{1}{2}(k+1)(k+2) - 1 - k) + 1$. Add $n$ vertices $\{w_1, w_2, \dots, w_n\}$, each adjacent to every $v_i$. Let $H_1, H_2, \dots, H_n$ be

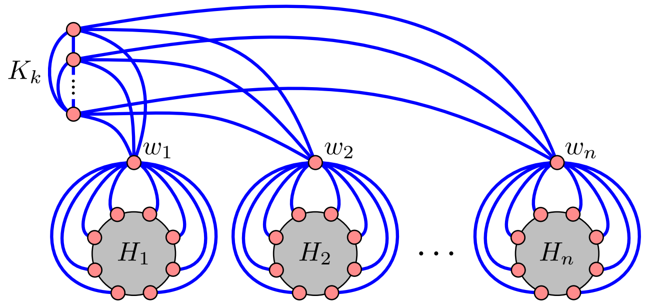


FIG. 7.3. *The graph $G_k$.*

copies of $G_{k-1}$. For all $1 \le j \le n$, add an edge between $w_j$ and each vertex of $H_j$, as illustrated in Figure 7.3. It is easily seen that from a tree decomposition of $G_{k-1}$ of width $k-1$ we can construct a tree decomposition of $G_k$ of width $k$. Thus $G_k$ has tree-width at most $k$.

To prove that $\mathtt{tn}(G_k) \ge \frac{1}{2}(k+1)(k+2)$, we proceed by induction on $k \ge 0$. Obviously $\mathtt{tn}(G_0) = 1$. Suppose that $\mathtt{tn}(G_{k-1}) \ge \frac{1}{2}k(k+1)$ but $\mathtt{tn}(G_k) \le \frac{1}{2}(k+1)(k+2)-1$. Since $\{v_1, v_2, \dots, v_k\}$ is a clique, we can assume that $v_i$ is in track $i$. Since each vertex $w_j$ is adjacent to each $v_i$, no $w_j$ is in tracks $\{1, 2, \dots, k\}$. There are $\frac{1}{2}(k+1)(k+2)-1-k$ remaining tracks. Since $n$ is more than twice this number, there are at least three $w_j$ vertices in a single track. Without loss of generality, $w_1 < w_2 < w_3$ in track $k+1$. No vertex $x$ of $H_2$ is in track $i \in \{1, 2, \dots, k\}$, as otherwise $xw_2$ would form an X-crossing with $v_iw_1$ or $v_iw_3$. No vertex $x$ of $H_2$ is in track $k+1$, since $x$ and $w_2$ are adjacent, and $w_2$ is in track $k+1$. Thus all the vertices of $H_2$ are in tracks $\{k+2, k+3, \dots, \frac{1}{2}(k+1)(k+2)-1\}$. There are $\frac{1}{2}(k+1)(k+2)-1-(k+1) = \frac{1}{2}k(k+1)-1$ such tracks. This contradicts the assumption that $\mathtt{tn}(G_{k-1}) \ge \frac{1}{2}k(k+1)$. Therefore $\mathtt{tn}(G_k) \ge \frac{1}{2}(k+1)(k+2)$.

It remains to prove that $\mathtt{tn}(G_k) \le \frac{1}{2}(k+1)(k+2)$. Suppose we have a $\frac{1}{2}k(k+1)$-track layout of $G_{k-1}$. Thus each $H_j$ has a $\frac{1}{2}k(k+1)$-track layout. Put each vertex $v_i$ of $G_k$ in track $i$. Put the vertices $\{w_1, w_2, \dots, w_n\}$ in track $k+1$ in this order. Put the track layout of each $H_j$ in tracks $k+2, k+3, \dots, \frac{1}{2}(k+1)(k+2)$ such that the vertices of $H_j$ precede the vertices of $H_{j+1}$. Clearly there are no X-crossings. ∎

Also note that Theorem 7.4 (for $k \ge 2$) can be extended using the proof technique of Lemma 2.3 to give the same lower bound for improper track layouts.

## 8. Open problems.

1. (In the conference version of their paper) Felsner, Liotta, and Wismath [42] asked whether every planar graph has a three-dimensional drawing with $\mathcal{O}(n)$ volume. By Theorem 2.9, this question has an affirmative answer if every planar graph has a $\mathcal{O}(1)$ track-number. Whether every planar graph has $\mathcal{O}(1)$ track-number is an open problem due to H. de Fraysseix [private communication, 2000] and, by Theorem 2.6, is equivalent to the following question.

2. Heath and colleagues [58, 54] asked whether every planar graph has a $\mathcal{O}(1)$ queue-number. The best known upper bound on the queue-number of a planar graph is $\mathcal{O}(\sqrt{n})$. In general, Dujmović and Wood [37] proved that every $m$-edge graph has queue-number at most $\boldsymbol{e}\sqrt{m}$, where $\boldsymbol{e}$ is the base of the natural logarithm.

3. Heath and colleagues [58, 54] also asked whether stack-number is bounded by queue-number (and vice-versa). Note that there is a family of graphs $\mathcal{G}$ with

$\mathsf{sn}(G) \in \Omega(3^{\Omega(\mathsf{qn}(G))-\epsilon})$ for all $G \in \mathcal{G}$ [54].

4. Is the queue-number of a graph bounded by a polynomial (or even singly exponential) function of its tree-width?

**Note added in proof.** Subsequent to this research, Dujmović and Wood [38] proved that graphs excluding a fixed graph as a minor, such as planar graphs, have three-dimensional drawings with $\mathcal{O}(n^{3/2})$ volume, as do graphs with bounded degree; Dujmović, Pór, and Wood [34] proved that track-number and queue-number are tied for all graphs; and Theorem 6.1 has been generalized (with a different proof) by Wood [92].

**Acknowledgments.** The authors are grateful for stimulating discussions with Prosenjit Bose, Jurek Czyzowicz, Hubert de Fraysseix, Stefan Langerman, Giuseppe Liotta, Patrice Ossona de Mendez, and Matthew Suderman. Thanks also to an anonymous referee for many helpful comments.

## REFERENCES

[1] M. O. Albertson and D. M. Berman, *Every planar graph has an acyclic* 7*-coloring*, Israel J. Math., 28 (1977), pp. 169–174.

[2] M. O. Albertson and D. M. Berman, *An acyclic analogue to Heawood's theorem*, Glasgow Math. J., 19 (1978), pp. 163–166.

[3] M. O. Albertson, G. G. Chappell, H. A. Kierstead, A. Kündgen, and R. Ramamurthi, *Coloring with no* 2*-colored* $P_4$*'s*, Electron. J. Combin., 11 (2004), paper R26.

[4] N. Alon and T. H. Marshall, *Homomorphisms of edge-colored graphs and Coxeter groups*, J. Algebraic Combin., 8 (1998), pp. 5–13.

[5] N. Alon, C. McDiarmid, and B. Reed, *Acyclic coloring of graphs*, Random Structures Algorithms, 2 (1991), pp. 277–288.

[6] N. Alon, B. Mohar, and D. P. Sanders, *On acyclic colorings of graphs on surfaces*, Israel J. Math., 94 (1996), pp. 273–283.

[7] S. Arnborg and A. Proskurowski, *Linear time algorithms for NP-hard problems restricted to partial $k$-trees*, Discrete Appl. Math., 23 (1989), pp. 11–24.

[8] F. R. Bernhart and P. C. Kainen, *The book thickness of a graph*, J. Combin. Theory Ser. B, 27 (1979), pp. 320–331.

[9] S. N. Bhatt, F. R. K. Chung, F. T. Leighton, and A. L. Rosenberg, *Scheduling tree-dags using FIFO queues: A control-memory trade-off*, J. Parallel Distrib. Comput., 33 (1996), pp. 55–68.

[10] H. L. Bodlaender, *A partial $k$-arboretum of graphs with bounded treewidth*, Theoret. Comput. Sci., 209 (1998), pp. 1–45.

[11] H. L. Bodlaender and J. Engelfriet, *Domino treewidth*, J. Algorithms, 24 (1997), pp. 94–123.

[12] O. V. Borodin, *On acyclic colorings of planar graphs*, Discrete Math., 25 (1979), pp. 211–236.

[13] O. V. Borodin, A. V. Kostochka, A. Raspaud, and E. Sopena, *Acyclic colouring of* 1*-planar graphs*, Discrete Appl. Math., 114 (2001), pp. 29–41.

[14] P. Bose, J. Czyzowicz, P. Morin, and D. R. Wood, *The maximum number of edges in a three-dimensional grid-drawing*, J. Graph Algorithms Appl., 8 (2004), pp. 21–26.

[15] I. Bruß and A. Frick, *Fast interactive* 3*-D graph visualization*, in Proceedings of the International Symposium on Graph Drawing (GD '95), F. J. Brandenburg, ed., Lecture Notes in Comput. Sci. 1027, Springer, New York, 1996, pp. 99–110.

[16] T. Calamoneri and A. Sterbini, 3*D straight-line grid drawing of* 4*-colorable graphs*, Inform. Process. Lett., 63 (1997), pp. 97–102.

[17] K. Chilakamarri, N. Dean, and M. Littman, *Three-dimensional Tutte embedding*, in Proceedings of the 26th Southeastern International Conference on Combinatorics, Graph Theory, and Computing, Congr. Numer. 107, 1995, pp. 129–140.

[18] M. Chrobak, M. Goodrich, and R. Tamassia, *Convex drawings of graphs in two and three dimensions*, in Proceedings of the 12th Annual ACM Symposium on Computational Geometry, 1996, ACM, New York, pp. 319–328.

[19] F. R. K. Chung, F. T. Leighton, and A. L. Rosenberg, *Embedding graphs in books: A layout problem with applications to VLSI design*, SIAM J. Algebraic Discrete Methods, 8 (1987), pp. 33–58.

[20] R. F. Cohen, P. Eades, T. Lin, and F. Ruskey, *Three-dimensional graph drawing*, Algorithmica, 17 (1996), pp. 199–208.

[21] I. F. Cruz and J. P. Twarog, *3D graph drawing with simulated annealing*, in Proceedings of the International Symposium on Graph Drawing (GD '95), F. J. Brandenburg, ed., Lecture Notes in Comput. Sci. 1027, Springer, New York, 1996, pp. 162–165.

[22] H. de Fraysseix, J. Pach, and R. Pollack, *How to draw a planar graph on a grid*, Combinatorica, 10 (1990), pp. 41–51.

[23] G. Di Battista, P. Eades, R. Tamassia, and I. G. Tollis, *Graph Drawing: Algorithms for the Visualization of Graphs*, Prentice-Hall, Englewood Cliffs, NJ, 1999.

[24] E. Di Giacomo, *Drawing series-parallel graphs on restricted integer 3D grids*, in Proceedings of the 11th International Symposium on Graph Drawing (GD '03), G. Liotta, ed., Lecture Notes in Comput. Sci. 2912, Springer, New York, pp. 238–246.

[25] E. Di Giacomo, G. Liotta, and H. Meijer, *3D Straight-Line Drawings of k-Trees*, Tech. Report 2003-473, School of Computing, Queen's University, Kingston, ON, Canada, 2003.

[26] E. Di Giacomo, G. Liotta, and S. Wismath, *Drawing series-parallel graphs on a box*, in Proceedings of the 14th Canadian Conference on Computational Geometry (CCCG '02), The University of Lethbridge, Lethbridge, AB, Canada, 2002, pp. 149–153.

[27] E. Di Giacomo and H. Meijer, *Track drawings of graphs with constant queue number*, in Proceedings of the 11th International Symposium on Graph Drawing (GD '03), G. Liotta, ed., Lecture Notes in Comput. Sci. 2912, Springer, New York, pp. 214–225.

[28] J. Díaz, J. Petit, and M. Serna, *A survey of graph layout problems*, ACM Comput. Surveys, 34 (2002), pp. 313–356.

[29] R. P. Dilworth, *A decomposition theorem for partially ordered sets*, Ann. of Math. (2), 51 (1950), pp. 161–166.

[30] G. Ding and B. Oporowski, *Some results on tree decomposition of graphs*, J. Graph Theory, 20 (1995), pp. 481–499.

[31] G. Ding and B. Oporowski, *On tree-partitions of graphs*, Discrete Math., 149 (1996), pp. 45–58.

[32] V. Dujmović, M. Fellows, M. Hallett, M. Kitching, G. Liotta, C. McCartin, N. Nishimura, P. Ragde, F. Rosemand, M. Suderman, S. Whitesides, and D. R. Wood, *On the parameterized complexity of layered graph drawing*, in Proceedings of the 5th Annual European Symposium on Algorithms (ESA '01), F. Meyer auf der Heide, ed., Lecture Notes in Comput. Sci. 2161, Springer, New York, 2001, pp. 488–499.

[33] V. Dujmović, P. Morin, and D. R. Wood, *Path-width and three-dimensional straight-line grid drawings of graphs*, in Proceedings of the 10th International Symposium on Graph Drawing (GD '02), M. T. Goodrich and S. G. Kobourov, eds., Lecture Notes in Comput. Sci. 2528, Springer, 2002, New York, pp. 42–53.

[34] V. Dujmović, A. Pór, and D. R. Wood, *Track layouts of graphs*, Discrete Math. Theor. Comput. Sci., 6 (2004), pp. 497–522.

[35] V. Dujmović and D. R. Wood, *Tree-Partitions of k-Trees with Applications in Graph Layout*, Tech. Report TR-2002-03, School of Computer Science, Carleton University, Ottawa, ON, Canada, 2002.

[36] V. Dujmović and D. R. Wood, *Tree-partitions of k-trees with applications in graph layout*, in Proceedings of the 29th Workshop on Graph Theoretic Concepts in Computer Science (WG'03), H. Bodlaender, ed., Lecture Notes in Comput. Sci. 2880, Springer, New York, 2003, pp. 205–217.

[37] V. Dujmović and D. R. Wood, *On linear layouts of graphs*, Discrete Math. Theor. Comput. Sci., 6 (2004), pp. 339–358.

[38] V. Dujmović and D. R. Wood, *Three-dimensional grid drawings with sub-quadratic volume*, in Towards a Theory of Geometric Graphs, J. Pach, ed., Contemp. Math. 342, AMS, 2004, pp. 55–66.

[39] P. Eades and P. Garvan, *Drawing stressed planar graphs in three dimensions*, in Proceedings of the International Symposium on Graph Drawing (GD '95), F. J. Brandenburg, ed., Lecture Notes in Comput. Sci. 1027, Springer, New York, 1996, pp. 212–223.

[40] P. Erdős, *Appendix,* in K. F. Roth*, On a problem of Heilbronn*, J. London Math. Soc., 26 (1951), pp. 198–204.

[41] S. Even and A. Itai, *Queues, stacks, and graphs*, in Proceedings of the International Symposium on Theory of Machines and Computations, Z. Kohavi and A. Paz, eds., Academic Press, New York, 1971, pp. 71–86.

[42] S. Felsner, G. Liotta, and S. Wismath, *Straight-line drawings on restricted integer grids in two and three dimensions*, J. Graph Algorithms Appl., 7 (2003), pp. 363–398.

[43] G. Fertin, A. Raspaud, and B. Reed, *Star coloring of graphs*, J. Graph Theory, 47 (2004), pp. 163–182.

[44] D. R. Fulkerson and O. A. Gross, *Incidence matrices and interval graphs*, Pacific J. Math., 15 (1965), pp. 835–855.

[45] J. L. Ganley and L. S. Heath, *The pagenumber of $k$-trees is $O(k)$*, Discrete Appl. Math., 109 (2001), pp. 215–221.

[46] M. R. Garey, D. S. Johnson, G. L. Miller, and C. H. Papadimitriou, *The complexity of coloring circular arcs and chords*, SIAM J. Algebraic Discrete Methods, 1 (1980), pp. 216–227.

[47] A. Garg, R. Tamassia, and P. Vocca, *Drawing with colors*, in Proceedings of the 4th Annual European Symposium on Algorithms (ESA '96), J. Diaz and M. Serna, eds., Lecture Notes in Comput. Sci. 1136, Springer, New York, 1996, pp. 12–26.

[48] M. C. Golumbic, *Algorithmic Graph Theory and Perfect Graphs*, Academic Press, New York, 1980.

[49] B. Grünbaum, *Acyclic colorings of planar graphs*, Israel J. Math., 14 (1973), pp. 390–408.

[50] R. Halin, *$S$-functions for graphs*, J. Geometry, 8 (1976), pp. 171–186.

[51] R. Halin, *Tree-partitions of infinite graphs*, Discrete Math., 97 (1991), pp. 203–217.

[52] F. Harary and A. Schwenk, *A new crossing number for bipartite graphs*, Utilitas Math., 1 (1972), pp. 203–209.

[53] T. Hasunuma, *Laying out iterated line digraphs using queues*, in Proceedings of the 11th International Symposium on Graph Drawing (GD '03), G. Liotta, ed., Lecture Notes in Comput. Sci. 2912, Springer, New York, 2004, pp. 202–213.

[54] L. S. Heath, F. T. Leighton, and A. L. Rosenberg, *Comparing queues and stacks as mechanisms for laying out graphs*, SIAM J. Discrete Math., 5 (1992), pp. 398–412.

[55] L. S. Heath and S. V. Pemmaraju, *Stack and queue layouts of posets*, SIAM J. Discrete Math., 10 (1997), pp. 599–625.

[56] L. S. Heath and S. V. Pemmaraju, *Stack and queue layouts of directed acyclic graphs. Part* II, SIAM J. Comput., 28 (1999), pp. 1588–1626.

[57] L. S. Heath, S. V. Pemmaraju, and A. N. Trenk, *Stack and queue layouts of directed acyclic graphs. Part* I, SIAM J. Comput., 28 (1999), pp. 1510–1539.

[58] L. S. Heath and A. L. Rosenberg, *Laying out graphs using queues*, SIAM J. Comput., 21 (1992), pp. 927–958.

[59] P. Hliněný, *Crossing-number critical graphs have bounded path-width*, J. Combin. Theory Ser. B, 88 (2003), pp. 347–367.

[60] S.-H. Hong, *Drawing graphs symmetrically in three dimensions*, in Proceedings of the 9th International Symposium on Graph Drawing (GD '01), P. Mutzel, M. Jünger, and S. Leipert, eds., Lecture Notes in Comput. Sci. 2265, Springer, New York, pp. 189–204.

[61] S.-H. Hong and P. Eades, *An algorithm for finding three dimensional symmetry in series parallel digraphs*, in Proceedings of the 11th International Conference on Algorithms and Computation (ISAAC '00), D. Lee and S.-H. Teng, eds., Lecture Notes in Comput. Sci. 1969, Springer, New York, 2000, pp. 266–277.

[62] S.-H. Hong and P. Eades, *Drawing trees symmetrically in three dimensions*, Algorithmica, 36 (2003), pp. 153–178.

[63] S.-H. Hong, P. Eades, A. Quigley, and S.-H. Lee, *Drawing algorithms for series-parallel digraphs in two and three dimensions*, in Proceedings of the 6th International Symposium on Graph Drawing (GD '98), S. Whitesides, ed., Lecture Notes in Comput. Sci. 1547, Springer, New York, 1998, pp. 198–209.

[64] M. Kaufmann and D. Wagner, eds., *Drawing Graphs: Methods and Models*, Lecture Notes in Comput. Sci. 2025, Springer, New York, 2001.

[65] A. V. Kostočka, *Acyclic* 6*-coloring of planar graphs*, Diskret. Analiz, 28 (1976), pp. 40–54.

[66] F. T. Leighton and A. L. Rosenberg, *Three-dimensional circuit layouts*, SIAM J. Comput., 15 (1986), pp. 793–813.

[67] J. Mitchem, *Every planar graph has an acyclic* 8*-coloring*, Duke Math. J., 41 (1974), pp. 177–181.

[68] B. Monien, F. Ramme, and H. Salmen, *A parallel simulated annealing algorithm for generating* 3*D layouts of undirected graphs*, in Proceedings of the International Symposium on Graph Drawing (GD '95), F. J. Brandenburg, ed., Lecture Notes in Comput. Sci. 1027, Springer, New York, 1996, pp. 396–408.

[69] J. Nešetřil and P. Ossona de Mendez, *Colorings and homomorphisms of minor closed classes*, in Discrete and Computational Geometry, The Goodman–Pollack Festschrift, B. Aronov, S. Basu, J. Pach, and M. Sharir, eds., Algorithms Combin. 25, Springer, New York, 2003, pp. 651–664.

[70] J. Nešetřil and A. Raspaud, *Colored homomorphisms of colored mixed graphs*, J. Combin. Theory Ser. B, 80 (2000), pp. 147–155.

[71] L. T. Ollmann, *On the book thicknesses of various graphs*, in Proceedings of the 4th Southeastern Conference on Combinatorics, Graph Theory and Computing, F. Hoffman, R. B. Levow, and R. S. D. Thomas, eds., Congr. Numer. 8, 1973, p. 459.

[72] D. I. Ostry, *Some Three-Dimensional Graph Drawing Algorithms*, Master's thesis, Department of Computer Science and Software Engineering, The University of Newcastle, Australia, 1996.

[73] J. Pach, T. Thiele, and G. Tóth, *Three-dimensional grid drawings of graphs*, in Advances in Discrete and Computational Geometry, B. Chazelle, J. E. Goodman, and R. Pollack, eds., Contemp. Math. 223, AMS, Providence, RI, 1999, pp. 251–255.

[74] S. V. Pemmaraju, *Exploring the Powers of Stacks and Queues via Graph Layouts*, Ph.D. thesis, Department of Computer Science, Virginia Polytechnic Institute and State University, Blacksburg, VA, 1992.

[75] Z. Peng, *Drawing Graphs of Bounded Treewidth/Pathwidth*, Master's thesis, Department of Computer Science, University of Auckland, New Zealand, 2001.

[76] T. Poranen, *A New Algorithm for Drawing Series-Parallel Digraphs in* 3*D*, Tech. Report A-2000-16, Department of Computer and Information Sciences, University of Tampere, Finland, 2000.

[77] B. A. Reed, *Algorithmic aspects of tree width*, in Recent Advances in Algorithms and Combinatorics, B. A. Reed and C. L. Sales, eds., Springer, New York, 2003, pp. 85–107.

[78] S. Rengarajan and C. E. Veni Madhavan, *Stack and queue number of* 2*-trees*, in Proceedings of the 1st Annual International Conference on Computing and Combinatorics (COCOON '95), D. Ding-Zhu and L. Ming, eds., Lecture Notes in Comput. Sci. 959, Springer, New York, 1995, pp. 203–212.

[79] N. Robertson and P. D. Seymour, *Graph minors.* II. *Algorithmic aspects of tree-width*, J. Algorithms, 7 (1986), pp. 309–322.

[80] D. J. Rose, R. E. Tarjan, and G. S. Lueker, *Algorithmic aspects of vertex elimination on graphs*, SIAM J. Comput., 5 (1976), pp. 266–283.

[81] P. Scheffler, *Die Baumweite von Graphen als ein Maß für die Kompliziertheit algorithmischer Probleme*, Ph.D. thesis, Karl-Weierstraß-Institut für Mathematik, Akademie der Wissenschaften der DDR, Berlin, Germany, 1989.

[82] W. Schnyder, *Planar graphs and poset dimension*, Order, 5 (1989), pp. 323–343.

[83] D. Seese, *Tree-partite graphs and the complexity of algorithms*, in Proceedings of the International Conference on Fundamentals of Computation Theory, L. Budach, ed., Lecture Notes in Comput. Sci. 199, Springer, New York, 1985, pp. 412–421.

[84] F. Shahrokhi and W. Shi, *On crossing sets, disjoint sets, and pagenumber*, J. Algorithms, 34 (2000), pp. 40–53.

[85] E. Steinitz, *Polyeder und Raumeinteilungen*, Encyclopädie Math. Wiss., 3AB12 (1922), pp. 1–139.

[86] R. Tarjan, *Sorting using networks of queues and stacks*, J. ACM, 19 (1972), pp. 341–346.

[87] M. Thorup, *All structured programs have small tree-width and good register allocation*, Information and Computation, 142 (1998), pp. 159–181.

[88] C. Ware and G. Franck, *Viewing a graph in a virtual reality display is three times as good as a* 2*D diagram*, in Proceedings of the IEEE Symposium on Visual Languages (VL '94), A. L. Ambler and T. D. Kimura, eds., IEEE Press, Piscataway, NJ, 1994, pp. 182–183.

[89] C. Ware and G. Franck, *Evaluating stereo and motion cues for visualizing information nets in three dimensions*, ACM Trans. Graphics, 15 (1996), pp. 121–140.

[90] C. Ware, D. Hui, and G. Franck, *Visualizing object oriented software in three dimensions*, in Proceedings of the IBM Centre for Advanced Studies Conference (CASCON '93), Toronto, 1993, pp. 1–11.

[91] D. R. Wood, *Queue layouts, tree-width, and three-dimensional graph drawing*, in Proceedings of the 22nd Foundations of Software Technology and Theoretical Computer Science (FST TCS '02), M. Agrawal and A. Seth, eds., Lecture Notes in Comput. Sci. 2556, Springer, New York, 2002, pp. 348–359.

[92] D. R. Wood, *Vertex partitions of chordal graphs*, submitted, 2004; see arXiv:math.CO/0408098.

[93] M. Yannakakis, *Embedding planar graphs in four pages*, J. Comput. System Sci., 38 (1986), pp. 36–67.

SIAM J. COMPUT. © 2005 Society for Industrial and Applied Mathematics
Vol. 34, No. 3, pp. 580–615

# ABSTRACT COMBINATORIAL PROGRAMS AND EFFICIENT PROPERTY TESTERS*

ARTUR CZUMAJ† AND CHRISTIAN SOHLER‡

**Abstract.** Property testing is a relaxation of classical decision problems which aims at distinguishing between functions having a predetermined property and functions being far from any function having the property. In this paper we present a novel framework for analyzing property testing algorithms. Our framework is based on a connection of property testing and a new class of problems which we call *abstract combinatorial programs*. We show that if the problem of testing a property can be reduced to an *abstract combinatorial program* of *small dimension*, then the property has an efficient tester.

We apply our framework to a variety of problems. We present efficient property testing algorithms for *geometric clustering* problems, for the *reversal distance* problem, and for *graph* and *hypergraph coloring* problems. We also prove that, informally, any *hereditary graph property* can be efficiently tested if and only if it can be reduced to an abstract combinatorial program of small size.

Our framework allows us to analyze all our testers in a unified way, and the obtained complexity bounds either match or improve the previously known bounds. Furthermore, even if the asymptotic complexity of the testers is not improved, the obtained proofs are significantly simpler than the previous ones. We believe that our framework will help to understand the structure of efficiently testable properties.

**Key words.** randomized algorithms, approximation algorithms, property testing, clustering problems, coloring, hereditary graph properties

**AMS subject classifications.** 68W20, 68W25, 68W40, 68Q25

**DOI.** 10.1137/S009753970444199X

## 1. Introduction.

In this paper, we consider *property testing* problems, that is, problems of determining whether a given function has a predetermined property or is "far" from any function having the property. A notion of property testing was first explicitly formulated by Rubinfeld and Sudan [69], where it was motivated mainly by its connection to program checking. The notion of property testing arises naturally in the context of program verification [15, 69], learning theory, and, in a more theoretical setting, probabilistically checkable proofs [9, 70], and it has been further developed in many follow-up works.

In [43], Goldreich, Goldwasser, and Ron initiated the study of property testing for *combinatorial objects*. They investigated several classical properties of labeled graphs and showed, for example, that $k$-colorability of graphs is testable in time independent of the input size. In this and other more recent papers (see, e.g., the excellent surveys in [34, 42, 68]), various algorithms have been proposed for testing graph and hypergraph properties [2, 4, 5, 7, 8, 23, 35, 43, 45, 46, 47, 56]; for testing geometric properties

*Received by the editors March 12, 2004; accepted for publication (in revised form) September 8, 2004; published electronically March 17, 2005. A preliminary version (extended abstract) of this paper appeared in *Proceedings of the 43rd Annual IEEE Symposium on Foundations of Computer Science (FOCS'02)*, IEEE Computer Society Press, Los Alamitos, CA, 2002, pp. 83–92. This research was supported in part by NSF grant CCR-0105701, by NSF ITR-grant CCR-0313219, by SBR grant 421090, by DFG grant Me 872-8/2, and by the IST program of the EU under contract IST-1999-14186 (ALCOM-FT).

http://www.siam.org/journals/sicomp/34-3/44199.html

†Department of Computer Science, New Jersey Institute of Technology, University Heights, Newark, NJ 07102-1982 (czumaj@cis.njit.edu).

‡Heinz Nixdorf Institute and Computer Science Department, University of Paderborn, D-33102 Paderborn, Germany (csohler@uni-paderborn.de).

[3, 24, 27, 28]; for testing properties of metrics [62] and matrices [37]; for testing properties of regular languages and branching problems [6, 38, 61]; for testing monotonicity [29, 36, 44], etc. (see, e.g., the result about quantum property testing [17]).

In all these algorithms the goal is to verify whether an input function (or an object) has a predetermined property or is "far" from having the property. Since the exact and deterministic solution to this problem may still be very hard to obtain, a property testing algorithm (*tester*) may have a one-sided error,[1] i.e., the tester must *accept* every function that has the property and must *reject* with probability at least $\frac{2}{3}$ every function that is "far" from having the property. To specify the notion of being "far" from having a property, we define a *distance* measure between functions. For a given parameter $\epsilon$, we say a *function is $\epsilon$-far from having a property if it has distance bigger than $\epsilon$ from any function having the property.*

Property testing may be seen as some kind of approximation algorithm for decision problems. Since we only want to "approximately decide" problems, it is often possible to obtain algorithms that are much more efficient than their exact counterparts. Many property testers have indeed a query complexity and running time that is sublinear in the input size. For some problems it is even possible to obtain property testers whose query complexity and running time is *independent* of the input size (see, e.g., the surveys [34, 42, 68]). This, in turn, resulted in the development of *sublinear*-time approximation algorithms for many classical combinatorial problems, including dense max-cut, clustering problems, and estimating the cost of the minimum spanning tree (see, e.g., [19, 20, 25, 26, 33, 39, 43, 52, 53]).

Even if there are known many very efficient testing algorithms, most of them have been analyzed using ad hoc techniques, designed specially for the problem at hand. There is still insufficient methodology and few tools that could help in the analysis of efficient testers for new problems. One such general approach has been developed by Goldreich, Goldwasser, and Ron [43], where a close relationship between property testing and PAC learning was investigated. In particular, it is shown that (in the two-sided error model) testing is not harder than (proper) learning, and thus known algorithms for learning can be applied in the context of property testing. Goldreich et al. [43] (see also Theorem 4.3 in [34]) presented also a fairly general framework (in the two-sided error model; see [47] for a characterization in a one-sided error model) of studying testing of certain graph partitioning problems. They were able to apply this framework to some graph problems, including graph coloring, clique, cut, and bisection. Another general approach, which uses the Szemerédi regularity lemma, has been proposed for studying graph problems and problems on matrices [4, 28, 39, 40, 56]. Even if this method is very powerful and has been shown to be very successful (for example, it allowed to prove that all first order graph properties without a quantifier alternation of type $\forall\exists$ have property testers whose complexity is independent of the size of the input graph), there are still many limitations of this approach. Furthermore, even though the bounds obtained by using the regularity lemma lead to the complexity bounds that are often independent from the input size, their dependence on the approximation parameter $\epsilon$ is often enormous (see the "tower" bounds in [4] and superpolynomial lower bounds in [2]).

**1.1. Framework for property testing.** The main contribution of this paper is a novel framework to analyze property testing algorithms. We consider *one-sided*

[1]In the literature a *two-sided error* model for property testing has also been investigated. In this paper we focus on the one-sided error model only.

*error* testers for properties of functions $f$ from a finite domain $\mathcal{D}$ to range $\mathcal{R}$. We denote by $f_{|X}$ the function $f$ restricted to $X$. We denote the tested property with $\Pi$. We consider property testing algorithms that choose a random sample of the domain $\mathcal{S} \subseteq \mathcal{D}$ and then verify whether $f$ agrees on $\mathcal{S}$ with some function $g$ having property $\Pi$. If this is the case, our sampling property tester accepts; otherwise, it rejects.

```
SAMPLING PROPERTY TESTER FOR PROPERTY $\Pi$.
    Sample a set $\mathcal{S}$ of $s$ objects from $\mathcal{D}$ uniformly at random
    **if** $f_{|\mathcal{S}} = g_{|\mathcal{S}}$ for some function $g$ with property $\Pi$ **then** *accept*
    **else** *reject*
```

Sampling property testers are simple to implement. The main difficulty with their use is the estimation of the sample size—what is the right sample size $s$ (the *query complexity* of the tester) so that the algorithm is a correct property tester? It is easy to see that if $f$ has the required property, then the algorithm will always accept $f$. Thus, the challenging part of the analysis is to estimate the value of $s$ such that if $f$ is far from having the property, then $f$ will be rejected with probability at least $\frac{2}{3}$. Our framework is designed to help in the analysis needed in this step.

In order to define our framework, we first introduce a notion of an *abstract combinatorial program* (ACP). An ACP is a triplet that consists of a *ground set*, which is typically a set of basic objects underlying the property testing problem; a set of *bases*, where each basis is a configuration of a subset of the ground set; and a *violation function* that verifies the input constraints and determines whether an element violates a given basis or not. We call a basis *feasible* if it is not violated by any object from the ground set. We investigate a generic problem of *testing feasibility of an ACP*. That is, for a given ACP, we want to distinguish whether the ACP has a feasible basis or any basis is violated by at least an $\epsilon$ fraction of objects from the ground set. Our main technical result is a theorem that gives a bound for the size of the random sample used in the sampling property tester for feasibility of ACPs. We show that if a certain monotonicity property is satisfied by an ACP, then the sample size $s$ can be chosen to be of size depending only on the maximum size of any basis in the ACP. Thus, the sample size is independent of the size of the ground set.

The main idea behind introducing ACPs is that for many properties ACPs capture the structure essential for testing the property. Therefore, in our framework we reduce property testing of a given property $\Pi$ to the problem of testing feasibility of related ACPs. We show that property $\Pi$ can be tested efficiently if there is a *reduction* to ACPs that satisfies two properties:

- the reduction is *distance preserving*, that is, any function that is far away from $\Pi$ is mapped to an ACP in which each basis is violated by many objects from the ground set; and
- the reduction is *feasibility preserving*, that is, if the function restricted to a sample set $S$ can be extended to a function with property $\Pi$, then there is a feasible basis for the subset of the ground set corresponding to $S$ in the ACP.

What are the advantages of our framework? We believe that our framework can be used as one of the first general tools to estimate the sample size for many tested properties. Our approach of using ACPs introduces/enforces a very important structure in the analysis of the properties and gives rather clear guidelines on how to perform the analysis. For many problems, our framework allows us to abstract the most important features of the problem at hand and to focus only on the combinatorial structures/properties of the problem. As a result, as we demonstrate in this paper,

Table 1.1
*Summary of specific results.*

| Problem | Source | Query complexity | Comments |
|---|---|---|---|
| $k$-radius clustering in $\mathbb{R}^d$ | Alon et al. [3] | $\widetilde{\mathcal{O}}(d \cdot k/\epsilon)$ | in any $L_p$ metric, $p \geq 1$ |
| | *this paper* | $\widetilde{\mathcal{O}}(d \cdot k/\epsilon)$ | in any $L_p$ metric, $p \geq 1$ |
| $k$-diameter clustering in $\mathbb{R}^d$ | Alon et al. [3] | $\Omega((1/\beta)^{(d-1)/4})$ | $\Omega(\sqrt{n})$ for $\beta = 0$ |
| | Alon et al. [3] | $\widetilde{\mathcal{O}}(k^2 \cdot d \cdot \epsilon^{-1} \cdot (2/\beta)^{2\,d})$ | in $L_2$ metric |
| | *this paper* | $\widetilde{\mathcal{O}}(k \cdot \epsilon^{-1} \cdot (1 + 2/\beta)^d)$ | in any $L_p$ metric, $p \geq 1$ |
| sorting by reversals | *this paper* | $\mathcal{O}(k/\epsilon)$ | |
| graph $k$-coloring | Goldreich et al. [43] | $\widetilde{\mathcal{O}}(k^4/\epsilon^6)$ | |
| | Alon and Krivelevich [7] | $\widetilde{\mathcal{O}}(k^2/\epsilon^4)$ | |
| | *this paper* | $\widetilde{\mathcal{O}}(k^2/\epsilon^4)$ | |
| $k$-coloring of $\ell$-uniform hypergraphs | Czumaj and Sohler [23] | $(\widetilde{\mathcal{O}}(k^2\,\ell^2/\epsilon^2))^\ell$ | |
| | Alon and Shapira [8] | $(\widetilde{\mathcal{O}}(k^{\ell-1}/\epsilon^2))^\ell$ | claimed $(\widetilde{\mathcal{O}}(k\,\ell/\epsilon^2))^\ell$ |
| | *this paper* | $(\widetilde{\mathcal{O}}(k\,\ell/\epsilon^2))^\ell$ | |

the use of this method often allows us to obtain short and elegant proofs.

Finally, we would like to mention that the notion of ACPs and the approach to correlate the algorithmic question of testing to the existence of a certain combinatorial structure is similar to the framework of LP-type problems introduced in [60].

**1.2. Applications of new framework: Specific results.** We demonstrate our framework on a variety of classical problems in computational geometry, graph algorithms, and computational biology (see Table 1.1 for a summary). We begin with two classical geometric clustering problems: *radius clustering* and *diameter clustering.* Then we discuss a standard problem in computational biology of *sorting by reversals*; we provide the first property tester for this problem. Next, we study labeled graph and hypergraph problems (in the adjacency matrix model). We first apply our approach to the *graph and hypergraph $k$-coloring problems.* Next, we consider testability of *hereditary graph properties* (that is, graph properties that are closed under taking induced subgraphs). We prove that any hereditary graph property can be tested with query complexity independent of the size of the input graph if and only if it can be reduced to an ACP in which each basis is of size independent of the size of the input graph. This result shows that for this specific and very important class of properties our framework captures the essence of property testing.

All our analyses of property testers listed above use our new framework of ACPs. All the obtained property testers have a *query complexity independent of the input size.* Furthermore, all our results either match or improve upon the best previously known query complexity of the problems; our tester for sorting by reversals is the first tester for this problem, and our result for hereditary graph properties is a new general tool to analyze graph properties. We will discuss now our specific results in more detail.

**1.2.1. Clustering.** We apply our framework to study two classical geometric clustering problems in $\mathbb{R}^d$: *radius clustering* [1, 3, 32], [51, p. 325] (also called the Euclidean $k$-center problem) and *diameter clustering* [3], [10, problem ND54], [41, problem MS9], [51, p. 326]. For a given point set $X$ in $\mathbb{R}^d$, the *radius* of $X$ is the radius of the smallest ball containing $X$, and the *diameter* of $X$ is the maximum distance between any two points in $X$. In this paper, we consider the problem under an arbitrary $L_p$ metric, $p \geq 1$. The (decision version of the) *radius clustering* problem is to decide if an input point set $P$ in $\mathbb{R}^d$ can be partitioned into $k$ clusters such that the radius of each cluster is bounded from above by a given real number $r$; the (decision version of the) *diameter clustering* problem is to decide if an input point set $P$ in $\mathbb{R}^d$ can be partitioned into $k$ clusters such that the diameter of each cluster is bounded from above by a given real number $b$.

It is well known that both the radius and the diameter clustering problems are $\mathcal{NP}$-complete and that it is $\mathcal{NP}$-hard to find a $(1+c)$-approximation algorithm for these two problems for certain positive constant $c$ [32] (for recent positive approximation results, see, e.g., [11, 49]). In the context of property testing, both radius and diameter clustering have been recently investigated by Alon et al. [3].

*Radius clustering.* For radius clustering under the $L_2$ metric, Alon et al. designed an $\epsilon$-tester having a query complexity of $\widetilde{\mathcal{O}}(d \cdot k/\epsilon)$. Given a positive real $r$, this tester accepts any point set $P$ in $\mathbb{R}^d$ that can be partitioned into $k$ clusters such that in each cluster all points can be enclosed by a ball of radius $r$. If $P$ cannot be partitioned in that way even if any $\epsilon$ fraction of the points is removed from $P$, then the tester rejects $P$ with probability at least $\frac{2}{3}$. Alon et al. [3] pointed out that their analysis can be extended to deal with the problem in arbitrary $L_p$ metrics, $p \geq 1$.

We apply our framework to design an $\epsilon$-tester for the radius clustering problem under any $L_p$ metric, $p \geq 1$. The query complexity of our tester is asymptotically the same as the query complexity of the tester from [3]. The main advantage of our approach is that it follows easily from our general framework (though the analysis from [3] is also relatively simple). We discuss our tester in detail here mostly to demonstrate our approach to a relatively simple but nontrivial problem.

*Diameter clustering.* For diameter clustering under the $L_2$ metric, Alon et al. [3] designed a so-called $(\epsilon, \beta)$*-tester* having a query complexity of $\widetilde{\mathcal{O}}(k^2 \cdot d \cdot \epsilon^{-1} \cdot (2/\beta)^{2\,d})$. Given $b > 0$, this tester accepts any point set $P$ in $\mathbb{R}^d$ that can be partitioned into $k$ clusters such that the diameter of every cluster is upper bounded by $b$. If, however, after removal of any $\epsilon$ fraction of the points in $P$ the obtained set cannot be partitioned into $k$ clusters so that the diameter of each cluster is upper bounded by $b \cdot (1+\beta)$, then the tester rejects $P$ with probability at least $\frac{2}{3}$. Alon et al. [3] also gave a lower bound for the query complexity of $(\epsilon, \beta)$-testers for diameter clustering, which is $\Omega(\sqrt{n})$ for $\beta = 0$ and $\Omega((1/\beta)^{(d-1)/4})$ for $\beta > 0$; these bounds hold for constant $\epsilon$ and $k = 1$.

Unlike in the radius clustering problem, Alon et al. [3] did not extend their results with arbitrarily small positive $\beta$ for testing diameter clustering to any other metric than $L_2$. For the values of $\beta$ around 1, they discussed the diameter clustering problem under an *arbitrary metric* (not necessarily $L_p$). On one hand, Alon et al. showed that there is an $(\epsilon, 1)$-tester having a query complexity of $\widetilde{\mathcal{O}}(k^2 \log k/\epsilon)$, and on the other hand, they showed that for any $\beta < 1$ every $(\epsilon, \beta)$-tester must have a query complexity of $\Omega(\sqrt{n/\epsilon})$.

For diameter clustering under the $L_2$ metric, we apply our framework to improve upon the result of Alon et al. [3] and design an $(\epsilon, \beta)$-tester having the query complexity of $\widetilde{\mathcal{O}}(k \cdot \epsilon^{-1} \cdot (1 + 2/\beta)^d)$. If $\beta \leq 1/d$, then we can obtain even a better bound for the query complexity of $\widetilde{\mathcal{O}}(k \cdot d \cdot \epsilon^{-1} \cdot (2/\beta)^{d-1})$. And so, in the most classical version of the diameter clustering problem on the plane, we improve the bound for the query complexity of Alon et al. [3] from $\widetilde{\mathcal{O}}(k^2\,(2/\beta)^4/\epsilon)$ to $\widetilde{\mathcal{O}}(k/(\beta\,\epsilon))$. Besides the improvement in the complexity of the property tester (quadratic with respect to $k$ and $1/\beta$), our analysis is significantly simpler than that in [3]. Moreover, unlike the method of Alon et al., our approach can be easily extended to deal with the diameter clustering problem under an arbitrary $L_p$ metric, $p \geq 1$.

**1.2.2. Reversals distance.** The study of genome comparisons and rearrangements is one of the major topics in modern molecular biology. Mathematical analysis of genome rearrangements was initiated by Sankoff, who introduced the *sorting by reversals problem* (see, e.g., [63, Chapter 10]). In sorting by reversals, one asks to compute the *reversal distance* of a given permutation, which is the minimum number of *reversals* (inversions of segments) needed to be performed to transform the permutation into the identity permutation. Because of its applications in computational biology, sorting by reversals has been widely studied in recent years (see, e.g., [12, 13, 18, 54, 63, 64]). For example, Pevzner and Waterman [64], in their collection of open problems in computational biology, mentioned algorithmic issues of the sorting by reversals problem as one of the most important problems in genome rearrangements. It is known that sorting by reversals is $\mathcal{NP}$-hard [18], that its optimization version is MAX-SNP-hard [14], and that there exits a polynomial-time 1.375-approximation algorithm [13] (see also [12, 54]).

In this paper, we introduce the notion of property testing in the context of sorting by reversals. We design a property testing algorithm that verifies whether a given permutation has reversal distance at most $k$ or is $\epsilon$-far from having reversal distance at most $k$. We apply our framework to show that this property tester has the query complexity of $\widetilde{\mathcal{O}}(k/\epsilon)$.

**1.2.3. Graphs and hypergraphs.** Our framework can be also successfully applied to test graph and hypergraph properties in the adjacency matrix representation model. We discuss here its applications to graph and hypergraph coloring and to testing arbitrary hereditary properties.

*Graph coloring.* The graph $k$-coloring problem is a classical problem in algorithmic graph theory. It is known that for $k \geq 3$ the problem of verifying whether an input graph is $k$-colorable is $\mathcal{NP}$-complete (see, e.g., [41, problem GT4] or [10, problem GT5]). It is also well known that this problem is very hard to approximate, and so, for example, it is $\mathcal{NP}$-hard to 4-color 3-colorable graphs, and it is hard to color $k$-colorable graphs with approximation within $n^{1-\epsilon}$, and even within $n^{1-\mathcal{O}(1/\sqrt{\log\log n})}$ [31]; here and in what follows, $n$ denotes the number of vertices in the graph. The best known approximation bound for arbitrary $k$ is of $\mathcal{O}(n\,(\log\log n)^2/\log^3 n)$ [10, problem GT5].

The problem of testing graph $k$-coloring has been first investigated in the seminal work of Goldreich et al. [43]. We say a graph is *$\epsilon$-far from $k$-colorable* if one has to remove more than $\epsilon\, n^2$ of its edges to obtain a $k$-colorable graph. Goldreich et al. [43] showed that it is possible to test with the query complexity of $\widetilde{\mathcal{O}}(k^4/\epsilon^6)$ whether a given graph has a $k$-coloring or is $\epsilon$-far from being $k$-colorable. This result has been improved to the query complexity of $\widetilde{\mathcal{O}}(k^2/\epsilon^4)$ by Alon and Krivelevich [7]. In the current paper, we present a simple analysis of a property tester based on our framework that matches the best known bounds from Alon and Krivelevich [7].

*Hypergraph coloring.* The simple structure of our framework allows us to extend relatively easily our analysis for graph coloring to obtain a very efficient *property tester for testing hypergraph coloring.* Let us recall that a *hypergraph* is a pair $\mathcal{H} = (V, E)$ with a finite vertex set $V$ and the edge set $E \subseteq 2^V$; we let $n$ denote the size of the vertex set $V$. A hypergraph $\mathcal{H}$ is *$\ell$-uniform* if $|e| = \ell$ for all $e \in E$; a 2-uniform hypergraph is a graph. A *$k$-coloring* of a hypergraph $\mathcal{H}$ is an assignment $\chi : V \to \{1, \ldots, k\}$. A coloring is *proper* if no edge in $E$ is *monochromatic*, that is, if for every edge $e \in E$, there are $v, u \in e$ with $\chi(v) \neq \chi(u)$. If $\mathcal{H}$ has a proper $k$-coloring, then $\mathcal{H}$

is *$k$-colorable*. The $k$-coloring problem for hypergraphs is to decide whether a given hypergraph is $k$-colorable. We say an $\ell$-uniform hypergraph is *$\epsilon$-far from $k$-colorable* if one has to remove more than $\epsilon n^\ell$ of its edges to obtain a $k$-colorable hypergraph.

Hypergraph coloring is a well-studied problem in the literature of discrete mathematics, combinatorics, and computer science. In contrast to graphs, where one can decide in linear time if a graph is 2-colorable (or, equivalently, bipartite), testing whether a given hypergraph is 2-colorable is $\mathcal{NP}$-complete even for 3-uniform hypergraphs [59]. In [57], it is shown that unless $\mathcal{NP} \subseteq \mathcal{ZPP}$, for any fixed $\ell \geq 3$, it is impossible to approximate in polynomial time the chromatic number of $\ell$-uniform hypergraphs within a factor $n^{1-\varepsilon}$ for any constant $\varepsilon > 0$. See also [48, 55] for further inapproximability results. The property of hypergraph 2-colorability has also been extensively studied in combinatorics (see, e.g., [21, 22, 30, 66]), and, for example, the study of this problem led to the discovery of the celebrated Lovász Local lemma [30].

In the context of property testing, Czumaj and Sohler [23] were the first to design efficient property testers for hypergraph coloring. They presented an $\epsilon$-tester for $k$-coloring $\ell$-uniform hypergraphs with the query complexity of $(\mathcal{O}(k^2\,\ell^2\,\log k/\epsilon^2))^\ell$. Alon and Shapira [8] applied a novel framework based on testing satisfiability of boolean formulas to design property testers for coloring uniform hypergraphs. The complexity of that tester is $\mathcal{O}((k^{\ell-1}\,\log k/\epsilon^2)^\ell)$, which is worse than that of Czumaj and Sohler. However, Alon and Shapira claimed they could modify the proof due to Czumaj and Sohler [23] to obtain an improvement in the query complexity to $\mathcal{O}((\ell\,k\,\log k/\epsilon^2)^\ell)$. In this paper, we show how our framework can be applied to test hypergraph coloring. The obtained property tester has asymptotically the same query complexity as the one claimed by Alon and Shapira [8] (both of these results have been obtained independently).

The generic results for testing hypergraph properties presented in [5] and [56] imply also new property testing algorithms for $k$-coloring $\ell$-uniform hypergraphs that achieve query complexity independent of the hypergraph size (that is, their query complexity is a function of $k$, $\ell$, and $\epsilon$ only). However, in these papers the dependence on $k$, $\ell$, and $\epsilon$ is significantly higher than in our paper.

*Hereditary graph properties.* Many interesting graph properties are *hereditary*; i.e., they are closed under taking induced graphs. And so, for example, being acyclic, stable (independent set), planar, perfect, bipartite, $k$-colorable, chordal, perfect, and having no induced predefined subgraph $K$ are all hereditary graph properties.

Hereditary graph properties have been extensively investigated in combinatorics, graph theory (see, e.g., [16] for a recent survey), and theoretical computer science (see, e.g., the classical hardness result in [58]). The class of hereditary graph properties contains also trivially all (decreasing) monotone graph properties (these are properties closed under edge removal). Monotone graph properties have been studied extensively in the literature, most notably in the context of the celebrated Aanderaa–Rosenberg conjecture (proven in [67]).

There are known hereditary graph properties (e.g., the graph $k$-coloring discussed above) being testable with the query complexity independent of the input graph size; such properties are often called *testable*. On the other hand, Goldreich and Trevisan [47] prove that there exist monotone (and thus hereditary) graph properties for which testing requires one to examine a constant fraction of the entries in the input graph adjacency matrix. It is a major problem in property testing to characterize graph properties, and in particular hereditary graph properties, that are testable. In this paper we prove that any hereditary graph property is testable with the query complexity independent of the input graph size if and only if it can be reduced to an ACP of constant size. This result implies that for hereditary graph properties our framework captures the entire essential structure of the property.

**1.3. Organization.** In section 2, we describe formally the property testing problem and then, in section 3, we introduce the concept of *abstract combinatorial programs.* In section 4, we discuss a (simple) version of our framework that reduces property testing to testing ACPs. In section 5, we apply our framework of testing ACPs to testing two classical *geometric clustering problems*: radius clustering and diameter clustering. Next, in section 6, we apply this framework to test the *reversal distances of permutations.* In section 7, we generalize our framework presented in section 4 and give a *full description of our framework.* In section 8, we apply this general framework to design a property tester for *$k$-coloring of graphs.* In section 9, we extend our testing analysis for graph coloring to design a property tester for *$k$-coloring of uniform hypergraphs.* Then, in section 10, we give a new *characterization of hereditary graph properties* and show that, essentially, any such property is efficiently testable if and only if it is efficiently testable in our framework. Finally, in section 11, we *summarize* the results obtained in the paper and provide final *conclusions.*

## 2. Preliminaries.

We begin with some basic notation and definitions. We use the $\widetilde{\mathcal{O}}$-notation to hide polylogarithmic factors; i.e., we have $\widetilde{\mathcal{O}}(n) = n \cdot \log^{\mathcal{O}(1)} n$. We write $[n] = \{1, \ldots, n\}$ to denote the set of integer numbers between 1 and $n$. Throughout this paper, we let $\mathcal{D}$ denote a finite set called *domain* and $\mathcal{R}$ a (possibly infinite) set called *range.* Next, let $\mathcal{F}$ denote a set of functions from $\mathcal{D}$ to $\mathcal{R}$. For a subset $S \subseteq \mathcal{D}$ and a function $f \in \mathcal{F}$ let $f_{|\,S}$ denote the *restriction* of $f$ to $S$. That is, $f_{|\,S} : S \to \mathcal{D}$ with $f_{|\,S}(x) = f(x)$ for all $x \in S$. Now we define a property of $\mathcal{F}$ as a set of functions from $\mathcal{F}$.

DEFINITION 2.1. *A set* $\Pi \subseteq \mathcal{F}$ *is called a* property.

As already mentioned in the introduction we also need a distance measure between functions in $\mathcal{F}$ to define a property testing problem. In general, such a distance measure can be an arbitrary function $\varsigma : \mathcal{F} \times \mathcal{F} \to [0, 1]$.

DEFINITION 2.2. *Given a distance measure $\varsigma$ between functions in $\mathcal{F}$ and a real number $\epsilon$, $0 \le \epsilon < 1$, we say a function $f \in \mathcal{F}$ is* $\epsilon$-far from (having a property) $\Pi$ *if $\varsigma(f, g) > \epsilon$ for every function $g \in \Pi$.*

Typically, we define the distance between two functions $f, g \in \mathcal{F}$ as the fraction of domain elements on which the two functions differ (see, for example, [43]). Therefore, we denote this distance measure as the *standard distance measure.*

DEFINITION 2.3. *Given two functions $f, g \in \mathcal{F}$ we define* the standard distance measure $\varsigma_1$ *for functions in $\mathcal{F}$ as*

$$\varsigma_1(f, g) \;=\; \frac{|\{x \in \mathcal{D} : f(x) \neq g(x)\}|}{|\mathcal{D}|} \quad .$$

The goal of property testing is to design efficient *property testers.* A property tester for $\Pi$ is an algorithm that takes as input a distance parameter $\epsilon$ and a (possibly implicit) description of $\mathcal{D}$. The property tester has *oracle access* to the input function $f$ (for each $x \in \mathcal{D}$ it may ask queries of the form, "What is the value of $f(x)$?"). A property tester must

- accept every function $f \in \Pi$; and
- reject every function $f$ that is $\epsilon$-far from $\Pi$ with probability at least $\frac{2}{3}$.[2]

[2]We consider a *one-sided error* model, though in the literature a *two-sided error* model has also

Notice that *if $f \notin \Pi$ and $f$ is not $\epsilon$-far from $\Pi$, then the outcome of the algorithm can go either way.*

*Complexity of property testers.* There are two types of possible complexity measures for property testers: the *query complexity* and the *running time.* In this paper we focus our attention on the query complexity, which measures the number of queries asked by a property testing algorithm.

DEFINITION 2.4. *The number of queries to the oracle is the* query complexity *of the property tester.*

## 3. Abstract combinatorial programs.

In this section we introduce *abstract combinatorial programs* (ACPs). An ACP is a combinatorial structure defined over a *ground set* of *atom items.* Some subsets of the ground set can describe possible "basic" configurations, and they are called the *bases* of the abstract combinatorial program. Furthermore, there is a relationship between atom items and bases: Each atom item is either *consistent* with a given basis or *violates* it; this relation is described by a *violation function.*

ACPs can be used to highlight combinatorial features of property testing problems. In typical applications the *ground set* depends on the problem under consideration and corresponds in a natural way to the basic items of the considered problem. For example, when we want to apply ACPs to graph problems, then the ground set might be the set of vertices of the graph, and when we consider properties of point sets, the ground set might be the set of points.

The set of *bases* describes possible "basic" configurations of the corresponding problem. If we consider a graph coloring problem, then a basis might correspond to a subset of vertices $W$ together with an associated coloring of $W$. For technical reasons we define every basis as a pair $(W, \ell)$, where $W$ is a subset of the ground set and $\ell$ is an index describing a configuration of $W$ (for example, in the graph coloring example above, it might encode a coloring of vertices in $W$).

The *violation function* describes for each basis $b$ and each atom item $x$ whether $x$ is consistent with $b$ or not. If $x$ is not consistent with $b$, then we say *$x$ violates $b$.* If every atom item is consistent with a basis, then we call this basis *feasible.* Normally, the violation function depends on the input instance. For the graph coloring example above one could define the violation function such that a vertex $v$ violates a basis (colored vertex set $W$) if and only if in the input graph the $k$-coloring of $W$ cannot be extended to a proper $k$-coloring of $W \cup \{v\}$.

Formally, we define an ACP in the following way.

DEFINITION 3.1. *An ACP is a triple $(\mathcal{C}, \mathcal{B}, \varpi)$, where*

- *$\mathcal{C}$ is a finite set called* ground set*;*
- *$\mathcal{B} \subseteq \{(W, \ell) : W \subseteq \mathcal{C}, \ell \in \mathbb{N}\}$ is a set of* bases*; and*
- *$\varpi : \mathcal{B} \to 2^{\mathcal{C}}$ is a* violation function.

*For each basis $b \in \mathcal{B}$ the set $\varpi(b)$ is the set of atom items violating $b$. A basis $b$ is* feasible *if $\varpi(b) = \emptyset$. An ACP is* feasible *if it has a feasible basis.*

Before we introduce some further definitions we give an example of how problem instances can be formulated as ACPs.

*Example* 1. We consider the problem of testing if a function $f : \mathbb{F} \to \mathbb{F}$ over a finite field $\mathbb{F}$ is a polynomial of degree at most $d$. We use the standard distance measure: a function is $\epsilon$-far from being a polynomial of degree at most $d$ if one has

been considered. In the two-sided error model the goal is to distinguish with probability at least $\frac{2}{3}$ between the case $f \in \Pi$ and the case of $f$ being $\epsilon$-far from $\Pi$.

to change more than $\epsilon \cdot |\mathbb{F}|$ function values to obtain a polynomial of degree no more than $d$.

In many cases (as in this example) one can formulate a problem instance as an ACP by using the domain of the tested functions as the ground set of the ACP. Consequently, in this example we have $\mathcal{C} = \mathbb{F}$. Bases typically correspond to "solutions" of the problem under consideration. In our example a basis should therefore correspond to a polynomial of degree at most $d$. By the fundamental theorem of algebra which states that a degree $d$ polynomial is uniquely defined by $d+1$ function values, we define the bases to be the set of $(d+1)$-tuples from $\mathcal{C}$. Thus, the set of bases is defined as

$$\mathcal{B} = \{(W, 1) : W \in \mathbb{F}^{d+1}\} \ .$$

We associate with each basis $(W, 1)$ the unique polynomial $p : \mathbb{F} \to \mathbb{F}$ with $p_{|W} = f_{|W}$. Then we define the violation function in a straightforward way: A basis $b = (W, 1)$ is violated by an atom item $c \in \mathcal{C}$ if $f(c) \neq p_b(c)$, where $p_b$ is the polynomial associated with $b$. Thus we get

$$\varpi(b) = \{c \in \mathcal{C} : f(c) \neq p_b(c)\} \ .$$

In this example we did not use the second parameter of the bases. This parameter comes into play when a single set of input constraints might correspond to multiple solutions of the problem. One such example can be found in section 5, where we consider clustering problems.

It is easy to see that the ACP in our example is feasible if and only if the corresponding function is a polynomial of degree at most $d$. Furthermore, if $f$ is $\epsilon$-far from being a polynomial of degree at most $d$, then every basis in the ACP is violated by more than $\epsilon \cdot |\mathbb{F}|$ atom items.

We are now interested in the problem of testing the feasibility of ACPs. Our property testing algorithm will explore properties of ACPs for subsets of the ground set, and for this we need further definitions.

DEFINITION 3.2 (ACP dimension and width). *Let $\mathcal{P}$ be an ACP. The* dimension *of $\mathcal{P}$,* $\dim(\mathcal{P})$, *is defined as* $\max\{|W| : (W, \ell) \in \mathcal{B}\}$. *The* width *of $\mathcal{P}$, width($\mathcal{P}$), is defined as* $\max\{\ell : (W, \ell) \in \mathcal{B}\}$.

DEFINITION 3.3 (self-feasible bases). *Let $\mathcal{P} = (\mathcal{C}, \mathcal{B}, \varpi)$ be an ACP. We say a basis $b = (W, \ell) \in \mathcal{B}$ is* covered *by a subset $C^* \subseteq \mathcal{C}$ if $W \subseteq C^*$. We say that a basis $b$ is* feasible *for a subset $C^* \subseteq \mathcal{C}$ if no $c \in C^*$ violates $b$. We say a subset $C^* \subseteq \mathcal{C}$* contains a self-feasible basis *if there is a basis $b$ that is covered by $C^*$ and that is feasible for $C^*$.*

In the next section, we design a property tester for feasibility of ACPs. We assume that the algorithm can determine for a set $S \subseteq \mathcal{C}$ if $S$ has a self-feasible basis or not. The size of the set $S$ should be as small as possible (the size of this set could be seen as the query complexity of the algorithm). To ensure that our property tester has a one-sided error, we consider only *monotone* ACPs.

DEFINITION 3.4 (monotonicity). *Let $\mathcal{P} = (\mathcal{C}, \mathcal{B}, \varpi)$ be an ACP with dimension* $\dim(\mathcal{P})$. *$\mathcal{P}$ is called* monotone *if it is either not feasible or if (it is feasible and) every subset $S \subseteq \mathcal{C}$ with $|S| \geq \dim(\mathcal{P})$ contains a self-feasible basis.*

## 3.1. Testing ACPs.

**3.1. Testing ACPs.** In this section we design a property tester for monotone ACPs. We first have to define when an ACP is far from feasible. We do this directly without specifying a distance measure between ACPs.

Definition 3.5. *An ACP is* $\epsilon$-far from feasible *if every basis is violated by more than $\epsilon \cdot |\mathcal{C}|$ objects from the ground set $\mathcal{C}$.*

A property tester for ACPs is an algorithm that (i) accepts every feasible ACP and (ii) rejects with probability at least $\frac{2}{3}$ every ACP that is $\epsilon$-far from feasible. The following is our main theorem, which establishes the quality of a simple property tester for ACPs.

Theorem 3.6 (testing ACPs). *Let $\mathcal{P} = (\mathcal{C}, \mathcal{B}, \varpi)$ be an ACP with dimension at most $\delta$ and width at most $\varrho$. Then there exists $s$ with*

$$s = \Theta(\epsilon^{-1} \cdot (\delta \cdot \ln(\delta/\epsilon) + \ln \varrho)) \ ,$$

*such that the algorithm*

```
ACP-TESTER(P, ε).
    Sample a set S of s objects from C uniformly at random
    **if** S contains a self-feasible basis **then** accept P
    **else** reject P
```

1. *accepts $\mathcal{P}$ if $\mathcal{P}$ is monotone and feasible, and*
2. *rejects $\mathcal{P}$ with probability at least $2/3$ if $\mathcal{P}$ is $\epsilon$-far from feasible.*

*Proof.* Let $\mathcal{P} = (\mathcal{C}, \mathcal{B}, \varpi)$ be an ACP that is $\epsilon$-far from feasible. Let $\mathcal{P}$ have dimension at most $\delta$ and width at most $\varrho$. For a basis $b = (W, \ell)$, let $\mathcal{E}_b$ be the random event (with respect to the random choice of $S$) that $W \subseteq S$ and that none of the elements from $\varpi(b)$ is in $S$. Now, in order to prove the theorem it is sufficient to show that with the probability larger than or equal to $\frac{2}{3}$ for none of $b \in \mathcal{B}$ the event $\mathcal{E}_b$ holds.

For every $r$, $0 \le r \le \delta$, let $\Delta_r$ be the set of all $b = (W, \ell) \in \mathcal{B}$ with $|W| = r$. Let us fix an arbitrary $b \in \Delta_r$. Then we have

$$\mathbf{Pr}[\mathcal{E}_b] \ = \ \frac{\binom{n-r-|\varpi(b)|}{s-r}}{\binom{n}{s}} \ \le \ \frac{\binom{(1-\epsilon)\,n-r}{s-r}}{\binom{n}{s}} \ .$$

Since $\mathcal{P}$ has dimension at most $\delta$ and width at most $\varrho$, we have $|\Delta_r| \le \varrho \cdot \binom{n}{r}$ for every $r \ge 0$. Furthermore, we have $|\Delta_r| = 0$ for all $r > \delta$. Therefore, by the union bound we obtain

$$\begin{aligned}
\mathbf{Pr}[\exists b \in \mathcal{B} : \mathcal{E}_b] \le \sum_{b \in \mathcal{B}} \mathbf{Pr}[\mathcal{E}_b] \ &= \ \sum_{r=0}^{\delta} \sum_{b \in \Delta_r} \mathbf{Pr}[\mathcal{E}_b] \ \le \ \varrho \cdot \sum_{r=0}^{\delta} \binom{n}{r} \cdot \frac{\binom{(1-\epsilon)\,n-r}{s-r}}{\binom{n}{s}} \\
&= \varrho \cdot \sum_{r=0}^{\delta} \binom{s}{r} \cdot \frac{((1-\epsilon)n - r) \cdots ((1-\epsilon)n - s + 1)}{(n-r) \cdots (n-s+1)} \\
&\le \varrho \cdot \sum_{r=0}^{\delta} s^r \cdot (1-\epsilon)^{s-r} \ \le \ \varrho \cdot \sum_{r=0}^{\delta} s^r \cdot e^{-\epsilon(s-r)} \\
&\le \varrho \cdot \delta \cdot s^{\delta} \cdot e^{-\epsilon(s-\delta)} \ \le \ \varrho \cdot \delta \cdot s^{\delta} \cdot e^{-\epsilon(s-s/2)} \ ,
\end{aligned}$$

where we assume in the last inequality that $s \ge 2\delta$. Then we set $s' = (\delta\, \epsilon^{-1}\, \ln(3\,\delta\,\varrho))^3$ and

$$s = 2\,\epsilon^{-1}\,(\delta\, \ln s' + \ln(3\,\delta\,\varrho)) \ .$$

With these choices, we have

$$\begin{aligned} s &= 2\,\epsilon^{-1}\cdot(\delta\,\ln s' + \ln(3\,\delta\,\varrho)) \;\le\; 2\,\epsilon^{-1}\,\delta\,\ln s'\,\ln(3\,\delta\,\varrho) \\ &\le 2\,(\epsilon^{-1}\,\delta\,\ln(3\,\delta\,\varrho))^2 \;\le\; (\delta\,\epsilon^{-1}\,\ln(3\,\delta\,\varrho))^3 \;=\; s' \enspace . \end{aligned}$$

We conclude that

$$\varrho\cdot\delta\cdot s^{\delta}\cdot e^{-\epsilon(s-s/2)} \;\le\; \varrho\cdot\delta\cdot s^{\delta}\cdot(s')^{-\delta}\cdot(3\delta\cdot\varrho)^{-1} \;\le\; 1/3 \enspace .$$

Hence, with probability at least $\frac{2}{3}$ all $b = (W, \ell) \in \mathcal{B}$ with $W \subseteq S$ are violated by $S$, which completes the proof of the first part of the theorem. If $\mathcal{P}$ is monotone and feasible, then every set $X \subseteq \mathcal{C}$ of size at least $\dim(\mathcal{P})$ contains a self-feasible basis. Therefore, $S$ must contain a self-feasible basis because $s \ge \dim(\mathcal{P})$. Hence the tester accepts the input. It remains to show that

$$s = \Theta(\epsilon^{-1}\cdot(\delta\cdot\ln(\delta/\epsilon) + \ln\varrho)) \enspace .$$

We have

$$\begin{aligned} s &= 2\,\epsilon^{-1}\,(\delta\,\ln s' + \ln(3\,\delta\,\varrho)) \;=\; \Theta(\epsilon^{-1}\,(\delta\,(\ln(\delta\,\epsilon^{-1}) + \ln\ln(\delta\,\varrho)) + \ln(\delta\,\varrho))) \\ &= \Theta(\epsilon^{-1}\,(\delta\,(\ln(\delta\,\epsilon^{-1}) + \ln\ln\varrho) + \ln\varrho)) \;=\; \Theta(\epsilon^{-1}\,(\delta\,\ln(\delta/\epsilon) + \ln\varrho)) \end{aligned}$$

by the observation that for $\delta \ge \sqrt{\ln\varrho}$ we have $\delta\ln\ln\varrho = \mathcal{O}(\delta\ln(\delta/\epsilon))$ and for $\delta < \sqrt{\ln\varrho}$ we have $\delta\ln\ln\varrho = o(\ln\varrho)$. □

COROLLARY 3.7. *Algorithm* ACP-TESTER *is a property tester for monotone ACPs.*

*Proof.* The proof follows immediately from Theorem 3.6. □

## 4. Property testing vs. testing ACPs.

Our motivation to introduce ACPs was to study their relation to property testing problems. We now prove a theorem that shows how we can use ACPs to prove for certain properties that there is an efficient property tester. Roughly speaking, a property can be tested with small query complexity if for every problem instance there is an equivalent (in the sense of property testing) ACP of small dimension and width.

We now present a first (simple) variant of Theorem 3.6. Then we give some examples and discuss in detail how our theorem can be used to prove the existence of a property tester with small query complexity. In most examples the obtained algorithm also has a small running time.

Our approach of using the framework of ACPs to study property testers of functions $f \in \mathcal{F}$ is to reduce testing of $f$ to testing certain ACPs. *In this section we consider only ACPs whose ground set $\mathcal{C}$ is the domain $\mathcal{D}$ of the function $f$.* Later, in section 7, we show how to deal with other, more general cases.

THEOREM 4.1. *Let $\mathcal{F}$ be a set of functions from a finite set $\mathcal{D}$ to a set $\mathcal{R}$ and let $\Pi$ be a property of $\mathcal{F}$. Let $0 < \epsilon < 1$ and let $\delta, \varrho \in \mathbb{N}$. If for every $f \in \mathcal{F}$ there exists an ACP $\mathcal{P}_f$ with $\dim(\mathcal{P}_f) \le \delta$ and width$(\mathcal{P}_f) \le \varrho$ such that*

- (distance preserving) *if $f$ is $\epsilon$-far from $\Pi$, then $\mathcal{P}_f$ is $\epsilon$-far from feasible; and*
- (feasibility preserving) *for every $X \subseteq \mathcal{C}$ with $|X| \ge \delta$: If there exists $g \in \Pi$ with $f_{|X} = g_{|X}$, then $X$ contains a self-feasible basis;*

*then there exists $s = \Theta(\epsilon^{-1}\cdot(\delta\cdot\ln(\delta/\epsilon) + \ln\varrho))$ such that the following algorithm is a property tester for property $\Pi$:*

```
TESTER($f, \epsilon$).
    Sample a set $S$ of $s$ elements in $\mathcal{D}$ uniformly at random
    if $f_{|S} = g_{|S}$ for some $g \in \Pi$ then accept $f$
    else reject $f$
```

*Proof.* In order to show that $\text{Tester}(f, \epsilon)$ is a property tester for $\Pi$, we have to prove that every function having property $\Pi$ is accepted by the tester, and every function that is $\epsilon$-far from having property $\Pi$ is rejected with probability at least $\frac{2}{3}$. If $f \in \Pi$, then for every $X \subseteq \mathcal{C}$ we have $f_{|X} = g_{|X}$ with $g = f \in \Pi$. This immediately implies that every $f \in \Pi$ is accepted by $\text{Tester}(f, \epsilon)$. Therefore, it remains to prove that if $f$ is $\epsilon$-far from $\Pi$, then the algorithm rejects the input with probability greater than or equal to $\frac{2}{3}$. We prove this by relating ACP-$\text{tester}(\mathcal{P}, \epsilon)$ to $\text{Tester}(f, \epsilon)$ and by applying Theorem 3.6.

By the distance preserving property, if $f$ is $\epsilon$-far from $\Pi$, then $\mathcal{P}_f$ is $\epsilon$-far from feasible. Furthermore, by Theorem 3.6, if $\mathcal{P}_f$ is $\epsilon$-far from feasible, then ACP-$\text{tester}(\mathcal{P}_f, \epsilon)$ rejects $\mathcal{P}_f$ with probability greater than or equal to $\frac{2}{3}$. $\mathcal{P}_f$ is rejected by ACP-$\text{Tester}(\mathcal{P}_f, \epsilon)$ only if the chosen sample set $S$ contains no self-feasible basis. But the feasibility preserving property implies that if there is $g \in \Pi$ that agrees with $f$ on the sample set, then every set $X \subseteq \mathcal{C}$ with $|X| \geq \delta \geq \dim(\mathcal{P}_f)$ contains a self-feasible basis. By the fact that $|S| \geq \delta$ we can conclude that $S$ contains a self-feasible basis if there exists a $g \in \Pi$ that agrees with $f$ on the sample set $S$. Therefore, we can conclude that if $f$ is $\epsilon$-far from $\Pi$, then with probability at least $\frac{2}{3}$ there is no such $g \in \Pi$ with $f_{|S} = g_{|S}$. Hence, $f$ is rejected by $\text{Tester}(f, \epsilon)$ with probability at least $\frac{2}{3}$. This implies that $\text{Tester}(f, \epsilon)$ is a property tester for $\Pi$. ☐

## 5. Clustering problems.

In this section, we apply our framework of testing ACPs to test two classical clustering problems. Clustering deals with the problem of partitioning a set of items into different groups called *clusters* such that a given optimization function is minimized. If we consider the corresponding decision problem, we want to know if a clustering with a given optimization value exists.

We consider two clustering problems of point sets in the $\mathbb{R}^d$: the *radius $k$-clustering* and the *diameter $k$-clustering.* Both problems have been analyzed in the context of property testing, and it is known that they possess efficient property testers [3]. Our goal is to show that these two problems can also be analyzed using our framework; in the case of the diameter clustering problem we also significantly improve the query complexity.

### 5.1. Radius clustering.

The decision version of the *radius $k$-clustering* problem in the Euclidean space $\mathbb{R}^d$ [3, 32], [51, p. 325] (sometimes also called the Euclidean $k$-center problem) consists of verifying whether a given set $P$ of $n$ points in $\mathbb{R}^d$ can be partitioned into $k$ sets such that the points in each set are contained in a unit ball. If such a partition exists, we say that $P$ is *$k$-clusterable.* We assume that $P$ is in a general position and that $P$ is represented as a function $f : [n] \to \mathbb{R}^d$, where $f(i)$ describes the location of the $i$th input point. Let $\mathcal{F}$ denote the set of functions representing point sets of size $n$ and let $\Pi \subseteq \mathcal{F}$ denote the set of functions representing point sets that are $k$-clusterable. The distance between two point sets is given by the standard distance measure between functions (see Definition 2.3), which is consistent with the following definition.

DEFINITION 5.1. *A set $P$ of $n$ points in $\mathbb{R}^d$ is* $\epsilon$-far from being $k$-clusterable *if more than $\epsilon n$ points must be deleted from $P$ to obtain a point set that is $k$-clusterable.*

In order to use our framework from Theorem 4.1 we have to describe for every input point set $P$ in $\mathbb{R}^d$ an ACP $\mathcal{P} = (\mathcal{C}, \mathcal{B}, \varpi)$ with $\mathcal{C} = [n]$ that satisfies the two conditions of the theorem. Let us first observe that the radius $k$-clustering property is a combinatorial property, that is, the identifiers of the points in the representation are irrelevant for the property. Thus, we can identify the ground set $\mathcal{C}$ of the ACP with the point set $P$. Hence, a basis of such ACP consists of a small set of points from $P$ (formally, of their corresponding indices) and some additional information (formally encoded as an integer).

We define the bases to specify all possible representations of feasible $k$-clusterings. For simplicity, let us first consider the radius 1-clustering problem. If the point set $P$ is 1-clusterable, then it is contained in some unit ball. Conversely, we can implicitly describe every "possible solution" of the problem by the position of such a unit ball. In a similar way, we can consider every ball with radius at most 1 as a possible solution. Our goal is to describe a subset of these balls implicitly in terms of atom items (points). If $P$ is 1-clusterable, then this subset must contain a ball that contains every point in $P$.

It is known that every finite point set $X$ is contained in a unique (closed) ball of smallest radius (see, e.g., [72]). We denote this ball by $\mathit{sball}(X)$. With every subset $W \subseteq P$ we associate its smallest enclosing ball $\mathit{sball}(W)$. If the radius of $\mathit{sball}(W)$ is at most 1, then this ball can be interpreted as a "possible solution" of the problem and $W$ (formally, the pair $(W, 1)$) as a basis. So we could say that for each $W \subseteq P$ the pair $(W, 1)$ is a basis if and only if the radius of $\mathit{sball}(W)$ is at most 1. But to obtain a query complexity independent of $n$ we need to reduce the number of atoms involved in a basis. To do this, we use the fact that there is always a subset $W \subseteq P$ of cardinality at most $d+1$ such that $\mathit{sball}(W) = \mathit{sball}(P)$ [72]. This motivates us to define $(W, 1)$ as a basis if the following two conditions are satisfied:

- $|W| \le d + 1$; and
- the radius of $\mathit{sball}(W)$ is at most 1.

We can now define a natural violation function in the following way: A basis $(W, 1)$ is violated by all points that are not contained in $\mathit{sball}(W)$. Using this definition we notice two important properties of our construction: (a) if $P$ is 1-clusterable, then $\mathcal{P}$ has a feasible basis, and (b) if $P$ is $\epsilon$-far from 1-clusterable, then every basis in $\mathcal{P}$ is violated by more than $\epsilon\, n$ points.

We can easily extend this definition of a basis to $k$ clusters: A basis for the radius $k$-clustering problem consists of $k$ bases for single clusters. Formally, a basis consists of at most $k\,(d+1)$ points from $P$ and an integer encoding a partition of the $k\,(d+1)$ points into $k$ sets (of size at most $d+1$). The violation function is defined in a straightforward way: a point violates a basis if it is not contained in any of the smallest enclosing balls defined by the bases.

*Bases for radius clustering.* We define the set of bases $\mathcal{B} = \{(W, \ell) : W \subseteq [n], |W| \le (d+1)\,k, 1 \le \ell \le k^{(d+1)\,k}\}$, where the pair $(W, \ell) \in \mathcal{B}$ should be interpreted as follows:

- $W$ is the set of points defining $k$ smallest enclosing balls (clusters); and
- $\ell$ is represented as a vector $\langle \nu_1, \dots, \nu_{(d+1)\,k} \rangle$ of length $(d+1)\,k$, where for $1 \le i \le k$, the $i$th point in $W$ is one of the points defining the smallest enclosing ball containing cluster number $\nu_i \in [k]$.

We say a basis $b \in \mathcal{B}$ is *valid* if, for every set $W'$ of points defining one of the $k$ smallest enclosing balls in $b$, the radius of $\mathit{sball}(W')$ is at most 1.

It is easy to see that with such a definition of the bases, the ACP designed for the radius clustering problem has $\dim(\mathcal{P}) = (d+1)\,k$ and $width(\mathcal{P}) = k^{(d+1)\,k}$.

*Violation function for radius clustering.* We say $p \in \mathcal{C} = [n]$ *violates a basis* $b \in \mathcal{B}$ if $b$ is not valid or the point $p$ (located at $f(p)$) is not contained in any of the $k$ balls defined by $b$. (Notice that all nonvalid bases are violated by all ground set elements.)

Once we have defined formally the ACP $\mathcal{P}$ for every instance of the radius clustering problem, we have to verify the prerequisites of Theorem 4.1: the distance preserving and the feasibility preserving properties.

*Distance preserving property.* If $P$ is $\epsilon$-far from being $k$-clusterable, then for every set of $k$ unit balls in $\mathbb{R}^d$ there is always a set of more than $\epsilon\, n$ points in $P$ that are not contained in any of the balls. By definition every basis corresponds to such a set of unit balls and the violation function is defined according to these balls. Thus every basis $b \in \mathcal{B}$ must be violated by more than $\epsilon\, n$ elements from $\mathcal{C}$. This implies the distance preserving property.

*Feasibility preserving property.* Every set $S$, $|S| \geq \delta$, that is $k$-clusterable is contained in $k$ unit balls. Thus we can partition $S$ into $k$ clusters $S_1, \dots, S_k$, each of which is contained in a unit ball. Furthermore, there exist sets $W_i \subseteq S_i$ with $sball(W_i) = sball(S_i)$ and $|W_i| \leq d+1$. The sets $W_i$ define a basis in $\mathcal{B}$ which is covered by $S$ and is feasible for $S$. This implies the feasibility preserving property.

*$L_p$ metrics.* We remark that the proof of [72] can be generalized to any $L_p$ metric. Hence, our analysis holds for any $L_p$ metric as well.

To summarize our discussion in this section, using our framework we can apply Theorem 4.1 to obtain a property tester for the radius clustering problem with a query complexity of $\widetilde{\mathcal{O}}(d\,k/\epsilon)$.

THEOREM 5.2. *There is a property tester for the radius clustering problem (in any $L_p$ metric) with query complexity of*

$$\mathcal{O}(d\,k\,\epsilon^{-1}\,\ln(d\,k/\epsilon)) = \widetilde{\mathcal{O}}(d\,k/\epsilon)\ .$$

**5.2. Diameter clustering.** The decision version of the diameter $k$-clustering problem (see [3], [10, problem ND54], [41, problem MS9], [51, p. 326]) is defined as follows: Given a point set $P$ in $\mathbb{R}^d$ and a positive integer $k$, can $P$ be partitioned into $k$ disjoint sets (clusters) $C_1, \dots, C_k$ such that for every $i$, $1 \leq i \leq k$, and every $x, y \in C_i$ it holds that $dist(x,y) \leq 1$? If such a partition exists, we say that $P$ is $k$-clusterable. As before, we assume in the property testing setting that the point set is represented by a function $f : [n] \to \mathbb{R}^d$.

Unlike for the radius clustering problem (and for all other problems discussed in this paper) we *do not use* the standard distance measure. This is because under the standard distance measure every property tester must have a query complexity of $\Omega(\sqrt{n})$ [3]. Therefore, we use instead the bicriteria distance measure that was proposed first in [3] and which is defined in the following way.

DEFINITION 5.3 (see [3]). *Let $P$ be a point set in $\mathbb{R}^d$ and $k$ a positive integer. We say $P$ is* $(\epsilon, \beta)$-far from being $k$-clusterable *if, for every partition of $P$ into sets $C_0, C_1, \dots, C_k$ satisfying $dist(x,y) \leq 1+\beta$ for all $1 \leq i \leq k$ and $x, y \in C_i$, it holds that $C_0 > \epsilon \cdot |P|$.*

It is known that under this distance measure there is a property tester with query complexity $\widetilde{\mathcal{O}}(k^2/\epsilon \cdot (2/\beta)^{2\,d})$ for the diameter $k$-clustering problem [3]. In this section we improve this result using our framework and decrease the query complexity to $\widetilde{\mathcal{O}}(k/\epsilon \cdot (2/\beta)^d)$. Our proof uses combinatorial arguments similar to those that appear implicitly in the proof presented in [3], but the use of our framework allows us to obtain a significant improvement in the complexity. Our goal is to design an efficient property tester that for given $k$, $\epsilon$ and $\beta > 0$, (i) accepts every point set that is $k$-clusterable and (ii) rejects with probability at least $\frac{2}{3}$ every input that is $(\epsilon, \beta)$-far from being $k$-clusterable. As in our analysis for the radius clustering problem, we denote by $\mathcal{F}$ all functions representing point sets of size $n$ and by $\Pi \subseteq \mathcal{F}$ all functions representing point sets that are $k$-clusterable.

We begin our discussion with the construction of bases for a single cluster. We use the following notation: A *cluster* is a nonempty set of points $C$ in $\mathbb{R}^d$ with $dist(x, y) \leq 1$ for every $x, y \in C$.

DEFINITION 5.4. *Let $C$ be a cluster. The* kernel $kern(C)$ of $C$ *is defined as the intersection of unit balls with centers at the points in $C$.*

We use the following simple properties of a kernel.

CLAIM 5.5. *Let $C$ be a cluster. Then we have the following:*

1. *$C \subseteq kern(C)$;*
2. *there exists a unit ball containing all the points in $C$; and*
3. *if $p \in kern(C)$, then $dist(x, y) \leq 1$ for every $x, y \in C \cup \{p\}$.*

*Proof.* (1) Since $dist(x, y) \leq 1$ for every $x, y \in C$, each point $x \in C$ is contained in every unit ball with the center at any other point $y \in C$, and hence $x$ is contained in $kern(C)$. (2) Since $C \neq \emptyset$, from the previous property we get $kern(C) \neq \emptyset$. Let us pick any point $x \in kern(C)$. Since $x$ is contained in all unit balls with centers at the points in $C$, it is at the distance at most 1 from every point in $C$. Therefore all points in $C$ are contained in the unit ball with the center at $x$. (3) If $p \in kern(C)$, then $p$ is contained in all unit balls with centers at the points in $C$, and thus its distance to every point in $X$ is at most 1. ◻

Now, in order to use our framework, from Theorem 4.1 we describe for every input set $P$ of $n$ points in $\mathbb{R}^d$ an ACP $\mathcal{P} = (\mathcal{C}, \mathcal{B}, \varpi)$ with $\mathcal{C} = [n]$ that satisfies the preconditions of the theorem. Following the arguments from the radius clustering case we identify the ground set $\mathcal{C}$ of the ACP with the input point set $P$. We first consider the case $k = 1$.

We start by making an observation about the kernel of a cluster $C$: for any point $p$, $C \cup \{p\}$ is a cluster if and only if $p$ is in the kernel of $C$. Thus we would like a good basis to contain exactly those elements from $C$ that define the "boundary" of $kern(C)$. Unfortunately, it might be that almost every element of $C$ defines the boundary of $C$, in which case the bases would be too large. Therefore, instead we find a small set of points $W$ in $C$ that approximates the kernel of $C$. We choose a basis for a cluster $C$ to be any maximal subset $W \subseteq C$ with the property that all points in $W$ have a mutual pairwise distance of at least $\beta$. In this way, we ensure that (a) the kernel is well approximated and (b) the number of points defining the kernel of a cluster is small.

Our first result shows that every basis for a single cluster is defined by at most $(1 + \frac{2}{\beta})^d$ points.

LEMMA 5.6. *Let $C$ be a cluster and let $W$ be a subset of $C$ such that for every $p, q \in W$ we have $dist(p, q) \geq \beta > 0$. Then $|W| \leq (1 + 2/\beta)^d$.*

*Proof.* Let $C$ and $W$ be as stated in the lemma. If we draw balls of radius $\beta/2$ centered at every point in $W$, then all these balls are pairwise disjoint. By property 2 in Claim 5.5, all points in $W$ are contained in some unit ball because $W \subseteq C$ and $C$ is a cluster. Therefore, if we draw balls of radius $\beta/2$ centered at every point in $W$, then all these balls are contained in a ball of radius $1 + \beta/2$. Since all these small balls are disjoint, we have the following upper bound for the size of $W$, where $Vol()$

denotes the volume of the object:

$$|W| \cdot Vol(\text{ball of radius } \beta/2) \ \le \ Vol(\text{ball of radius } 1+\beta/2) \ .$$

Since the volume of a $d$-dimensional ball of radius $r$ is equal to[3] $\frac{r^d \cdot \pi^{d/2}}{\Gamma(1+d/2)}$, we obtain an upper bound of $(1+(2/\beta))^d$ for the size of $W$. ◻

To formalize our definition of bases we need the following.

DEFINITION 5.7. *Let $P$ be a point set in $\mathbb{R}^d$, and let $\beta$ be a positive real. Let $C$ be a cluster. We say a point $p \in P$ is* $\beta$-covered *by $C$ if $p \in kern(C)$, and there is $q \in C$ such that $dist(p,q) \le \beta$.*

We say a pair $b = (W, \ell)$ is a *basis* for the diameter 1-clustering problem if $W$ is a subset of $P$ of size at most $(1+\frac{2}{\beta})^d$ and $\ell = 1$. We say a basis $(W,\ell)$ is *valid* if for all $p, q \in W$ we have $\beta < dist(p,q) \le 1$. It remains to define the violation function. If a basis is not valid, it is violated by every point in $P$. If a basis is valid, we have two different types of violation: First, a point $p$ violates a basis $b = (W,1)$ if $p \notin kern(W)$. This is the straightforward type of violation. If a point is not in the kernel of $W$, then it cannot belong to the same cluster. The second type of violation is different. A point $p \in kern(W)$ violates $b$ if $p$ is not $\beta$-covered by $W$. The intuition behind this definition is such that a point violates a basis if it is consistent with the current basis of the cluster, but it changes the kernel significantly, and hence it is no longer a good implicit description of the solution of the clustering problem.

*Bases for diameter clustering.* We can extend our definition of bases for the diameter 1-clustering problem to arbitrary $k$ in the following way: A basis for the diameter $k$-clustering problem is an encoding of $k$ sets $W_1, \dots, W_k \subseteq P$ each of size at most $(1+\frac{2}{\beta})^d$. A basis is *valid* if for every $p, q \in W_i$, $1 \le i \le k$, it holds that $\beta < dist(p,q) \le 1$. Formally, a basis is a set $W = \bigcup_i W_i$ with an integer encoding the partition of $W$ in the sets $W_i$.

LEMMA 5.8. *For any point set in $\mathbb{R}^d$, the ACP $\mathcal{P}$ for the diameter $k$-clustering problem has dimension at most $k \cdot (1+2/\beta)^d$ and width at most $k^{k\cdot(1+2/\beta)^d}$.*

*Proof.* The dimension of the ACP follows immediately from the definition of the bases. The width follows from the fact that every point of a basis can belong to one of $k$ sets; that is, we have (at most) $k$ choices for each point of the basis. ◻

*Violation function for diameter clustering.* A basis $b$ that is an encoding of the sets $W_1, \dots, W_k$ is violated by a point $p \in P$ if $b$ is not valid or if $p$ violates every $W_i$ (seen as a basis for the 1-clustering problem), $1 \le i \le k$.

*Feasibility preserving property.* In order to show the feasibility preserving property we have to show that every $k$-clusterable set $S \subseteq P$ of size at least $\delta = k \cdot (1+2/\beta)^d \ge \dim(\mathcal{P})$ has a self-feasible basis. If $S$ is $k$-clusterable, then there exists a partition of $S$ into $k$ clusters $C_1, \dots, C_k$. Since for every $W_i \subseteq C_i$ it holds that $kern(C_i) \subseteq kern(W_i)$, we know that there exist sets $W_i$ with the property that for each $p, q \in W_i$ we have $\beta < dist(p,q) \le 1$ and for each $p \in C_i$ and $q \in W_i$ we have $dist(p,q) \le \beta$. By Lemma 5.6 and the bound of the size of the bases, the feasibility preserving property follows.

*Distance preserving property.* We prove the distance preserving property by contradiction. Let us assume $P$ is $(\epsilon,\beta)$-far from being $k$-clusterable, and suppose there is a basis $b$ encoding the sets $W_1, \dots, W_k$ that is violated by less than $\epsilon\, n$ points. We

[3]Here, $\Gamma()$ is Euler's Gamma (factorial) function, defined as $\Gamma(x) = \int_0^\infty t^{x-1}\, e^{-t}\, dt$ for all positive $x$. It is well known that $\Gamma(x+1) = x\,\Gamma(x)$ and that for integer $x \ge 0$ we have $\Gamma(x+1) = x!$ and $\Gamma(x+\frac{1}{2})! = \sqrt{\pi} \cdot ((2\,x)!)/(x! \cdot 4^x)$.

delete all points in $P$ that violate $b$ and let $P^*$ be the remaining point set. Since all the points in $P^*$ are $\beta$-covered by some $W_i$, for each point $p \in P^*$ there is a $W_i$ with $p \in kern(W_i)$ and for which there exists $q_p \in W_i$ with $dist(p, q_p) \leq \beta$. We assign each such point $p$ to the cluster corresponding to $W_i$. Observe that all points in the cluster are contained in $kern(W_i)$. Furthermore, for every point $r \in kern(W_i)$ the distance between $p$ and $r$ is not larger than the distance from $p$ to $q_p$ plus the distance from $q_p$ to $r$. Hence, we can conclude that the distance between two points in the cluster (both of which must be contained in $kern(W_i)$) is at most $1 + \beta$. This implies that $P^*$ can be partitioned into $k$ clusters of diameter at most $1 + \beta$ each, which is a contradiction.

**5.2.1. Generalization to $L_p$ metrics.** It is not hard to see that our entire analysis carries over to arbitrary $L_p$ metrics. Indeed, all our arguments but the volume argument in the proof of Lemma 5.6 are independent of the metric space. The volume arguments in the proof of Lemma 5.6 can be easily extended to an arbitrary $L_p$ metric by observing that the volume of a ball with radius $r$ in the $d$-dimensional $L_p$ space is equal to $r^d \cdot (2\,\Gamma(1 + 1/p))^d / \Gamma(1 + d/p)$; see, e.g., [65, p. 11]. Using this bound in the proof of Lemma 5.6, one can easily see that also that the lemma is true for arbitrary $L_p$ metrics.

Now, we can summarize our discussion above and apply Theorem 4.1 to obtain the following result.

THEOREM 5.9. *There is a property tester for the diameter $k$-clustering problem in an arbitrary $L_p$ metric with the query complexity of*

$$\widetilde{\mathcal{O}}(k \cdot \epsilon^{-1} \cdot (1 + (2/\beta))^d) \ .$$

**6. Reversal distance.** In this section we consider the *reversal distance problem.* In sorting by reversals one is asked to compute the shortest sequence of (interval) *reversals* that transforms a given permutation $\pi$ into the identity permutation. The number of reversals that are necessary is called the *reversal distance* between $\pi$ and the identity permutation.

We now introduce the problem formally. Let $\mathbb{S}_n$ denote the set of all permutations of $[n]$.

DEFINITION 6.1. *A reversal $\varrho\langle i, j\rangle$ of an interval $[i, j]$, $1 \leq i \leq j \leq n$, is the permutation*

$$\begin{pmatrix} 1 & 2 & \dots & i-1 & i & i+1 & \dots & j-1 & j & j+1 & \dots & n \\ 1 & 2 & \dots & i-1 & j & j-1 & \dots & i+1 & i & j+1 & \dots & n \end{pmatrix}.$$

*That is, for a permutation $\pi = (\pi_1, \dots, \pi_n) \in \mathbb{S}_n$, $\varrho\langle i, j\rangle$ has the effect of reversing the order of $(\pi_i, \pi_{i+1}, \dots, \pi_j)$ and transforming $\pi$ into*

$$\pi \cdot \varrho\langle i, j\rangle \ = \ (\pi_1, \dots, \pi_{i-1}, \pi_j, \pi_{j-1}, \dots, \pi_i, \pi_{j+1}, \dots, \pi_n) \ .$$

We now define the reversal distance between two permutations.

DEFINITION 6.2. *Given a pair of permutations $\pi = (\pi_1, \dots, \pi_n), \sigma = (\sigma_1, \dots, \sigma_n) \in \mathbb{S}_n$, the* reversal distance *$d_{rev}(\pi, \sigma)$ between $\pi$ and $\sigma$ is the minimum number of reversals needed to transform $\pi$ into $\sigma$ (that is, the minimum number $k$ such that there exists a sequence of reversals $\varrho_1, \varrho_2, \dots, \varrho_k$ with $\pi \cdot \varrho_1 \cdot \varrho_2 \cdots \varrho_k = \sigma$).*

Equivalently, we can compute the number of reversals necessary to transform $\sigma^{-1}\pi$ into the identity permutation $id$. Therefore, we are interested in the reversal distance between a given permutation $\pi$ and the identity permutation. We consider the decision version of the reversal distance problem.

DEFINITION 6.3. *The* reversal distance *problem consists of determining, for a given permutation $\pi \in \mathbb{S}_n$ and an integer $k$, whether the reversal distance $d_{rev}(\pi, id)$ between $\pi$ and the identity permutation $id$ is at most $k$.*

We now formulate the problem as a property testing problem that fits into our framework. The set of functions $\mathcal{F}$ we want to consider is the set of permutations of $[n]$, that is, $\mathcal{F} = \mathbb{S}_n$. We are interested in all permutations that have a reversal distance of at most $k$ to the identity permutation. We can write this as a property $\Pi$ as follows:

$$\Pi = \{\pi \in \mathbb{S}_n : d_{rev}(\pi, id) \le k\} \ .$$

We use the standard distance measure from Definition 2.3, which is equivalent to the following.

DEFINITION 6.4. *A permutation $\pi \in \mathbb{S}_n$ is $\epsilon$-far from having reversal distance smaller than or equal to $k$ if for every sequence of $k$ reversals $\varrho_1, \varrho_2, \ldots, \varrho_k$ the permutation $\pi \cdot \varrho_1 \cdot \varrho_2 \cdots \varrho_k$ disagrees with the identity permutation on more than $\epsilon \cdot n$ places; that is, if $\pi \cdot \varrho_1 \cdot \varrho_2 \cdots \varrho_k = (\sigma_1, \ldots, \sigma_n)$, then $|\{i \in \{1, 2, \ldots, n\} : \sigma_i \ne i\}| > \epsilon \cdot n$.*

In contrast to the clustering problems, the reversal distance property is not combinatorial. Therefore, we have to take the domain of the function into account. For a permutation $\pi = (\pi_1, \ldots, \pi_n)$ we denote atom items by $\pi_i$. This notion covers the fact that the value of domain element $i$ is $\pi_i = \pi(i)$. Hence it captures also the domain of a value of $f$.

Let us notice that we can encode an interval $[i, j]$ by the two domain elements $\pi_i$ and $\pi_j$ (using the fact that $\pi^{-1}(\pi_i) = i$ and $\pi^{-1}(\pi_j) = j$). If we apply a reversal $\varrho$ to $\pi$, then $\pi_i$ and $\pi_j$ *induce* the interval $\big[((\pi \cdot \varrho)^{-1})(\pi_i), ((\pi \cdot \varrho)^{-1})(\pi_j)\big]$ (for this reason we want to work with $\pi_i$ rather than with $i$). We denote the interval induced by two elements $\pi_i$ and $\pi_j$ by $[\pi_i, \pi_j]$.

We say a reversal $\varrho\langle r, s\rangle$ *splits* an interval $[\pi_i, \pi_j]$ if $i < r \le j$ or $i \le s < j$ (or both).

DEFINITION 6.5. *Let $\pi = (\pi_1, \ldots, \pi_n)$ be a permutation and let $[\pi_i, \pi_j]$ denote an interval. We say that a reversal $\varrho\langle k, \ell\rangle$* splits *an interval $[\pi_i, \pi_j]$ if $i < k \le j$ or if $i \le \ell < j$.*

We generalize this notion to $k$-reversals.

DEFINITION 6.6. *Let $\pi = (\pi_1, \ldots, \pi_n)$ be a permutation. A $k$-reversal $\varrho = \varrho_1 \cdot \varrho_2 \cdots \varrho_k$* splits *an interval $[\pi_i, \pi_j]$ if there exists $\ell$, $0 \le \ell < k$, such that $\varrho_{\ell+1}$ splits*

$$[(\pi \cdot \varrho_1 \cdots \varrho_\ell)^{-1}(\pi_i), (\pi \cdot \varrho_1 \cdots \varrho_\ell)^{-1}(\pi_j)] \ .$$

*If $\varrho$ does not split $[\pi_i, \pi_j]$, then we say $\varrho$ is* safe *for $[\pi_i, \pi_j]$.*

Notice that if $\varrho_1, \ldots, \varrho_k$ is *safe* for $[\pi_i, \pi_j]$, then each of the reversals $\varrho_1, \ldots, \varrho_k$ either entirely contains $[\pi_i, \pi_j]$ or it does not contain any $\pi_\ell \in [\pi_i, \pi_j]$. Therefore, in this case, after applying $\varrho_1, \ldots, \varrho_k$ the positions of $\pi_{i+1}, \ldots, \pi_{j-1}$ are determined by the position of $\pi_i$ and $\pi_j$.

*Bases for the $k$-reversal problem.* We define a basis as a set of $2k + 1$ intervals induced by pairs of the atom items of the basis. For each such set we consider only reversals that are safe for these intervals. We say a set of $2k + 1$ intervals is a basis if there exists a sequence of $k$ reversals $\varrho_1, \ldots, \varrho_k$ such that for each atom item $\pi_i$ of the basis $(\pi \cdot \varrho_1 \cdots \varrho_k)^{-1}(\pi_i) = \pi_i$ and if $\varrho_1, \ldots, \varrho_k$ is safe for all intervals induced by the elements involved in the basis.

DEFINITION 6.7. *Let $\pi = (\pi_1, \dots, \pi_n) \in \mathbb{S}_n$. A set $\mathcal{I}$ of $2k+1$ intervals is a valid basis for the reversal distance problem if there is a sequence $\varrho_1, \dots, \varrho_k$ of $k$ reversals such that*

- *$(\pi \cdot \varrho_1 \cdots \varrho_k)^{-1}(\pi_i) = \pi_i$ and $(\pi \cdot \varrho_1 \cdots \varrho_k)^{-1}(\pi_j) = \pi_j$ for each interval $[\pi_i, \pi_j] \in \mathcal{I}$; and*
- *no interval $[\pi_i, \pi_j] \in I$ is split by $\varrho_1, \dots, \varrho_k$.*

*If the set of intervals is a basis $b$, then we associate with it the $k$-reversal $\varrho_b = \varrho_1 \cdots \varrho_k$. (In case that there are different sequences that witness the basis property we choose an arbitrary one.)*

Formally, a basis consists of $4k+2$ atom items and an integer number encoding the $2k+1$ pairs of atom items (an atom item may be paired with itself). The integer number can be seen as a vector of length $2k+1$ having entries with values from $[4k+2] \times [4k+2]$ to specify each of the $2k+1$ pairs.

LEMMA 6.8. *For each instance of the reversal distance problem the corresponding ACP $\mathcal{P}$ has $\dim(\mathcal{P}) \leq 4k+2$ and $width(\mathcal{P}) \leq (4k+2)^{4k+2}$.*

*Proof.* By definition, a basis consists of $4k+2$ atom items. Thus we have $\dim(\mathcal{P}) \leq 4k+2$. Each vector of length $2k+1$ with values from $[4k+2] \times [4k+2]$ can be encoded as an integer number between 1 and $(4k+2)^{4k+2}$. Thus we have $width(\mathcal{P}) \leq (4k+2)^{4k+2}$. ◻

*Violation function for k-reversal distance.* Let $b$ be a basis and let $\varrho_b = \varrho_1 \cdots \varrho_k$ be the $k$-reversal associated with basis $b$. We say $b$ is violated by $\pi_i \in \mathcal{C}$ if $(\pi \cdot \varrho_b)^{-1}(\pi_i) \neq \pi_i$; that is, $\pi_i$ is not moved to position $\pi_i$ when $\varrho_b$ is applied to $\pi$.

*Distance preserving property.* We have associated a $k$-reversal $\varrho_b$ to each basis. An atom item violates a basis if it is not at the correct position when $\varrho_b$ is applied to $\pi$. If a permutation is $\epsilon$-far from having reversal distance smaller than or equal to $k$, then every $k$-reversal puts more than $\epsilon n$ elements to the wrong position. Hence every basis is violated by more than $\epsilon\, n$ elements. Therefore, the distance preserving property is satisfied.

*Feasibility preserving property.* Let $S \subseteq \mathcal{C}$ be a set of atom items and let $\varrho = \varrho_1 \cdots \varrho_k$ be a $k$-reversal with $(\pi \cdot \varrho)^{-1}(\pi_i) = \pi_i$ for each $\pi_i \in S$. We show that in this case $S$ has a self-feasible basis. First we want to construct a set of $2k+1$ intervals that are safe for $\varrho_1, \dots, \varrho_k$. We start with the set of $s-1$ intervals induced by $S$. Now we observe that each reversal $\varrho_i$ can split at most 2 of these intervals. We conclude that at most $2k$ of these intervals are split by $\varrho_1, \dots, \varrho_k$. We can merge adjacent intervals not split by $\varrho_1, \dots, \varrho_k$ and obtain a set of $2k+1$ intervals that are not split by $\varrho_1, \dots, \varrho_k$. Hence there exists a sequence of $k$ reversals that is safe for each of our intervals. Thus these intervals form a basis $b$. It remains to prove that this basis is not violated (the $k$-reversal associated with the basis must not be the $k$-reversal $\varrho$). By our construction of the intervals each $\pi_i \in S$ is contained in a safe interval. Therefore, its position after applying the reversal is uniquely determined by the positions of the endpoints of the interval. Let $S_I \subseteq S$ denote the set of endpoints of intervals of the basis $b$. Since $b$ is a basis there is a $k$-reversal $\varrho_b$ with

$$(\pi \cdot \varrho_b)^{-1}(\pi_i) \;=\; \pi_i \;=\; (\pi \cdot \varrho)^{-1}(\pi_i) \text{ for each } \pi_i \in S_I \;\;.$$

Since the endpoints are mapped to the same places when $\varrho_b$ and $\varrho$ are applied to $\pi$, we can conclude that each other point in $S$ is also mapped to the same place. Hence no $\pi_i \in S$ violates $b$, and we have shown the feasibility preserving property.

Once we have proven the distance and the feasibility preserving properties, Lemma 6.8 and Theorem 4.1 allow us to conclude with the following theorem.

Theorem 6.9. *There exists a property tester for the $k$-reversal distance property with query complexity $\widetilde{\mathcal{O}}(k/\epsilon)$.*

## 7. Property testing vs. testing ACPs: General case.

In section 4, we described a framework for testing problems via testing ACPs. We only considered ACPs whose ground set is identical to the domain of the tested function. Although we showed that this framework can be applied to various problems, it is not always powerful enough to deal with a larger spectrum of problems. In some cases it is necessary to consider ground sets different from the domain of the tested function. For example, if we consider the adjacency matrix model using the approach from section 4 and represent a graph by a function $f : V \times V \to \{0,1\}$, then the ground set would be a set of entries in the adjacency matrix. But sampling entries of the adjacency matrix results in a disconnected set of edges if the size of the sample set is $o(\sqrt{n})$ [43]. For graph problems, typically a better approach would be to identify the ground set with the set of vertices of the graph and then analyze the subgraph induced by these vertices. To make a more flexible model in order to deal with problems more complex than this one, we introduce *interpretations.*

*Interpretations.* Interpretations are functions that map each subset of the ground set of the ACP to a subset of the domain of the tested function. Given a sample set $S$ of ground set items, we use the interpretation to determine a set of domain elements $\mathcal{D}_S$. Then we query for the value $f(x)$ for each $x \in \mathcal{D}_S$.

Definition 7.1. *An* interpretation *of $\mathcal{C}$ in $\mathcal{D}$ is a function $I : 2^{\mathcal{C}} \to 2^{\mathcal{D}}$.*

To investigate quantitative properties of the reduction, we need the following definition.

Definition 7.2. *For a function $h : \mathbb{N} \to \mathbb{N}$, we say an interpretation $I$ of $\mathcal{C}$ in $\mathcal{D}$ is $h$-*bounded *if for every $X \subseteq \mathcal{C}$ it holds that $|I(X)| \le h(|X|)$. (We write in that case that $I$ is $h(N)$-bounded, with $N$ being the formal input variable.)*

The main idea behind introducing these notions is to allow a more general analysis of algorithm Tester$(f,\epsilon)$ from section 4. Similarly to the proof of Theorem 4.1, we want to test an input function $f \in \mathcal{F}$ via testing a related ACP $\mathcal{P} = (\mathcal{C}, \mathcal{B}, \varpi)$. Since $\mathcal{P}$ is now allowed to be an ACP with an arbitrary ground set $\mathcal{C}$, we use the interpretation $I$ of $\mathcal{C}$ in $\mathcal{D}$ to link the domains of $f$ and $\mathcal{P}$ in the reduction. The notion of $h$-bounded functions in Definition 7.2 is used to describe the size of the random sample in the tester. That is, if the interpretation $I$ is $h(N)$-bounded and if our algorithm samples a set $S \subseteq \mathcal{C}$, then we query for the value of $f(x)$ for every $x \in I(S) \subset \mathcal{D}$, and the restriction $|I(S)| \le h(s)$ yields an upper bound on the query complexity.

*Distance preserving property.* In Theorem 4.1 we used the distance preserving property that requires that if a function $f$ is $\epsilon$-far from property $\Pi$, then the ACP is $\epsilon$-far from feasible. In general, however, one can parameterize this property and require the *$(\epsilon,\lambda)$-distance preserving* property: *if $f$ is $\epsilon$-far from property* $\Pi$*, then the ACP is $\lambda$-far from feasible.*

Now, in the framework defined above, it is easy to see that Theorem 4.1 can be generalized to the following theorem, which describes our framework in its full generality.

Theorem 7.3. *Let $\mathcal{F}$ be a set of functions from a finite set $\mathcal{D}$ to a set $\mathcal{R}$, and let $\Pi$ be a property of $\mathcal{F}$. Let $0 < \epsilon, \lambda < 1$ and let $I : 2^{\mathcal{C}} \to 2^{\mathcal{D}}$ be an $h$-bounded interpretation of $\mathcal{C}$ in $\mathcal{D}$. If for every $f \in \mathcal{F}$ there exists an ACP $\mathcal{P}_f$ with $\dim(\mathcal{P}_f) \le \delta$ and width$(\mathcal{P}_f) \le \varrho$ such that*

- ($(\epsilon,\lambda)$-distance preserving) *if $f$ is $\epsilon$-far from $\Pi$, then every basis in $\mathcal{P}_f$ is $\lambda$-far from feasible; and*

- (feasibility preserving) *for every $X \subseteq \mathcal{C}$ with $|X| \geq \delta$: If there exists $g \in \Pi$ with $f_{|I(X)} = g_{|I(X)}$, then $X$ contains a self-feasible basis;*

*then there exists $s = \Theta(\lambda^{-1} \cdot (\delta \cdot \ln(\delta/\lambda) + \ln \varrho))$ such that the following algorithm is a property tester for $\Pi$ with query complexity $h(s)$:*

```
TESTER($f, \epsilon$).
    Sample a set $S$ of $s$ elements in $\mathcal{C}$ uniformly at random
    if $f_{|I(S)} = g_{I(S)}$ for some $g \in \Pi$ then accept $f$
    else reject $f$
```

*Proof.* In order to show that $\text{Tester}(f, \epsilon)$ is a property tester for $\Pi$, we have to prove that every function having property $\Pi$ is accepted by the tester, and every function that is $\epsilon$-far from having property $\Pi$ is rejected with probability at least $\frac{2}{3}$. If $f \in \Pi$, then for every $X \subseteq \mathcal{C}$ we have $f_{|I(X)} = g_{|I(X)}$ with $g = f \in \Pi$. This immediately implies that every $f \in \Pi$ is accepted by $\text{Tester}(f, \epsilon)$. Therefore, it remains to prove that if $f$ is $\epsilon$-far from $\Pi$, then the algorithm rejects the input with probability greater than or equal to $\frac{2}{3}$. We prove this by relating ACP-tester$(\mathcal{P}, \lambda)$ to $\text{Tester}(f, \epsilon)$ and by applying Theorem 3.6.

By the distance preserving property, if $f$ is $\epsilon$-far from $\Pi$, then $\mathcal{P}_f$ is $\lambda$-far from feasible. Furthermore, by Theorem 3.6, if $\mathcal{P}_f$ is $\lambda$-far from feasible, then ACP-tester$(\mathcal{P}_f, \lambda)$ rejects $\mathcal{P}_f$ with probability greater than or equal to $\frac{2}{3}$. $\mathcal{P}_f$ is rejected by ACP-tester$(\mathcal{P}_f, \lambda)$ only if the chosen sample set $S$ contains no self-feasible basis. But now the feasibility preserving property implies that if there is $g \in \Pi$ that agrees with $f$ on the interpretation of the sample set, then every set $X \subseteq \mathcal{C}$ with $|X| \geq \delta \geq \dim(\mathcal{P}_f)$ contains a self-feasible basis. By the fact that $|S| \geq \delta$ we can conclude that $S$ contains a self-feasible basis if there exists a $g \in \Pi$ that agrees with $f$ on the sample set $S$. Therefore, we can conclude that if $f$ is $\epsilon$-far from $\Pi$, then with probability at least $\frac{2}{3}$ there is no such $g \in \Pi$ with $f_{|I(S)} = g_{|I(S)}$. Hence, $f$ is rejected by $\text{Tester}(f, \epsilon)$ with probability at least $\frac{2}{3}$. This implies that $\text{Tester}(f, \epsilon)$ is a property tester for $\Pi$. □

## 8. Graph coloring.

In this section we apply Theorem 7.3 to graph coloring. A $k$-coloring of a graph $G = (V, E)$ is an assignment $\chi : V \to \{1, \ldots, k\}$ of colors to the vertices of the graph. A coloring is *proper* if there is no edge $e = (v, u) \in E$ such that $\chi(v) = \chi(u)$. If $G$ has a proper $k$-coloring, then $G$ is *$k$-colorable.* The graph $k$-coloring problem consists of determining whether a given graph is $k$-colorable.

We consider the graph coloring problem in the *adjacency matrix model.* That is, the input graph $G = (V, E)$ is given as a function $V \times V \to \{0, 1\}$ representing the adjacency matrix of the graph. Therefore we have $f(u, v) = 1$ if and only if $(u, v) \in E$. Without loss of generality, we assume that $V = [n]$. Let $\Pi$ denote all $n$-vertex graphs that have a proper $k$-coloring. We use the standard distance measure between graphs (see Definition 2.3), which is equivalent to the following definition.

Definition 8.1. *A graph $G$ is* $\epsilon$-far from being $k$-colorable *if in order to transform $G$ into a $k$-colorable graph one has to modify more than $\epsilon n^2$ entries in the adjacency matrix of $G$.*

It is known that graph coloring in the adjacency model can be tested efficiently [43]. The main contribution of our framework is a clear and elegant proof that highlights the combinatorial aspects of the problem and a slight improvement in the query complexity that matches for $k > 2$ the bounds from [7] in the $\widetilde{\mathcal{O}}$-notation.

For the $k$-coloring problem, we identify the ground set $\mathcal{C}$ with the set of vertices $V$ of the input graph $G = (V, E)$. Since in our framework $G$ can be viewed as given by its adjacency matrix representation, we define the interpretation $I$ to map each set of vertices to the submatrix induced by these vertices. That is, for every $W \subseteq V$, we have $I(W) = W \times W$. Clearly, the interpretation is $N^2$-bounded.

The bases of the coloring problem are formed by some properly colored sets of vertices. That is, every basis corresponds to a pair $(W, \chi^*)$, where $W \subseteq V = \mathcal{C}$ and $\chi^*$ is an encoding of a proper $k$-coloring of $W$ (here, one can think that a $k$-coloring of $W$ is represented by a vector of length $|W|$ with values in $k^W$). Notice that with this definition, if for each $(W, \chi^*) \in \mathcal{B}$ we have $|W| \leq \delta$, then the so-defined ACP $\mathcal{P}$ has $\dim(\mathcal{P}) \leq \delta$ and $\mathit{width}(\mathcal{P}) \leq k^{\delta}$.

Before we define the bases formally, we need some more definitions.

Definition 8.2. *Let $S \subseteq V$ be a set of vertices and let $\chi$ be a proper $k$-coloring of $S$. Let*

$$V_i \;=\; \bigl\{v \in V : \exists u \in S \textit{ with } \chi(u) = i \textit{ and } (v,u) \in E\bigr\}$$

*be the set of vertices that cannot be properly colored in color $i$ using any extension of $\chi$ to $V$. We call a vertex $v \in V \setminus S$* heavy *for $\langle S, \chi\rangle$ if there is a proper extension of $\chi$ to $S \cup \{v\}$, but every extension increases the number of vertices in certain $V_i$, $1 \leq i \leq k$, by at least $\epsilon n/3$ (that is,* (i) *there exists $1 \leq j \leq k$, with $v \notin V_j$, and* (ii) *$\forall_{1\leq j\leq k}$ if $v \notin V_j$, then $\bigl|\{w \notin V_j : (v,w) \in E\}\bigr| \geq \epsilon n/3$).*

*A vertex $v \in V \setminus S$ that cannot be properly colored by any extension of the coloring $\chi$ (that is, $v \in \bigcap_{i=1}^{k} V_i$) is called a* conflict *vertex for $\langle S, \chi\rangle$.*

*Bases for $k$-coloring.* For the graph coloring problem we define the bases inductively.

- $\{\emptyset, 1\}$ is a basis (where 1 is the encoding of the coloring of the empty set of vertices).
- If $b = (K, \chi)$ is a basis, $v$ is a *heavy* vertex for $b$, and $\chi^*$ is an encoding of the previous coloring $\chi$ of $K$ extended by a proper coloring of $v$, then $(K \cup \{v\}, \chi^*)$ is a basis.

Our next step is to show that the ACP has small dimension and width.

Lemma 8.3. *For every input instance of the graph coloring problem the corresponding ACP $\mathcal{P}$ has* $\dim(\mathcal{P}) \leq 3k/\epsilon$ *and* $\mathit{width}(\mathcal{P}) \leq k^{3k/\epsilon}$.

*Proof.* Since every $k$-coloring of a set of $r$ vertices can be encoded using an integer number between 1 and $k^r$, it is enough to show that $|K| \leq 3k/\epsilon$ for every basis $b = (K, \chi)$.

Let $b = (K, \chi)$ be a basis with $|K| = r$. Since $b$ is a basis, there must exist a sequence of bases $(K_0, \chi_0)$, $(K_1, \chi_1)$, $\dots$, $(K_r, \chi_r)$ with $(K_0, \chi_0) = (\emptyset, 1)$, $(K_r, \chi_r) = (K, \chi)$, and such that for every $1 \leq i \leq r$ we have $|K_i| = i$ and the only vertex in $K_i \setminus K_{i-1}$ is heavy for $\langle K_{i-1}, \chi_{i-1}\rangle$.

Let $V_1^{(i)}, V_2^{(i)}, \dots, V_k^{(i)}$, $0 \leq i \leq r$, be the sets of vertices such that each vertex $v \in V_j^{(i)}$ cannot be properly colored in color $j$ if we want to extend coloring $\chi_i$ to the set $K_i \cup \{v\}$. That is,

$$V_j^{(i)} \;=\; \bigl\{v \in V : \exists u \in K_i \text{ with } \chi_i(u) = j \text{ and } (v,u) \in E\bigr\}\ .$$

It is easy to see that for every $i, j$ we have $V_j^{(i)} \subseteq V_j^{(i+1)}$. Furthermore, by the definition of heavy vertices, we know that for every $i$, $1 \leq i \leq r$, there is certain $j$, $1 \leq j \leq k$, such that $|V_j^{(i)}| \geq |V_j^{(i-1)}| + \epsilon n/3$. Therefore, since we have $V_j^{(0)} = \emptyset$ for every $j$, $1 \leq j \leq k$, it must hold that $\sum_{j=1}^{k} |V_j^{(i)}| \geq i\,\epsilon n/3$ for every $i$, $1 \leq i \leq r$. Finally, since $|V_j^{(r)}| \leq n$ for every $j$, $1 \leq j \leq k$, we conclude that $r \leq 3k/\epsilon$. ◻

*Violation function for $k$-coloring. A basis $b = (K, \chi)$ is* violated *by a vertex $v \in V$ if either* (i) *$v$ is a heavy vertex for $\langle K, \chi \rangle$ or* (ii) *$v$ is a conflict vertex for $\langle K, \chi \rangle$.*

Once we have described $k$-coloring in our framework, in order to apply Theorem 7.3, we must show that the $(\epsilon, \lambda)$-distance preserving and the feasibility preserving properties hold. We begin with the proof of the following lemma that implies the $(\epsilon, \lambda)$-distance preserving property with $\lambda = \epsilon/3$.

LEMMA 8.4 ($(\epsilon, \epsilon/3)$-distance preserving property). *Let $G = (V, E)$ be a graph that is $\epsilon$-far from being $k$-colorable and let $S \subseteq V$ be any set of properly $k$-colored vertices with a proper coloring $\chi$. Then $V$ contains more than $\epsilon n/3$ conflict vertices for $\langle S, \chi \rangle$ or $V$ has more than $\epsilon n/3$ heavy vertices for $\langle S, \chi \rangle$.*

*Proof.* Our proof is by contradiction. Assume $G$ is $\epsilon$-far from having a proper $k$-coloring and there are less than $\epsilon n/3$ conflict vertices for $\langle S, \chi \rangle$ and less than $\epsilon n/3$ heavy vertices for $\langle S, \chi \rangle$. Then we show that there is a $k$-colorable graph $G^*$ that is obtained from $G$ by removing less than $\epsilon n^2$ edges. This would yield a contradiction and hence would conclude the proof.

Let $X$ be the set of all conflict vertices for $\langle S, \chi \rangle$, let $Y$ be the set of all heavy vertices for $\langle S, \chi \rangle$, and let $Z$ be the set of remaining uncolored vertices (i.e., $Z = V \setminus (S \cup X \cup Y)$). For every $i$, $1 \le i \le k$, let $V_i$ be the set of vertices in $V$ such that for every $v \in V_i$ the extension of $\chi$ by coloring $v$ with color $i$ is not a proper coloring (cf. Definition 8.2).

We first construct a graph $G'$ by removing all edges incident to the vertices in $X \cup Y$ and extend the coloring $\chi$ of $S$ to a $k$-coloring of $S \cup X \cup Y$ by coloring the vertices in $X \cup Y$ arbitrarily. Since $|X \cup Y| < 2\epsilon n/3$, less than $2\epsilon n^2/3$ edges are removed from $G$ in this way. Furthermore, since all vertices in $X \cup Y$ are isolated, the obtained coloring $\chi'$ is a proper $k$-coloring of $S \cup X \cup Y$.

Now, we modify $G'$ to extend the coloring $\chi'$ to all vertices in $Z$. For each $v \in Z$, let $\tau(v)$ be a color that satisfies the following two constraints:

- If we extend $\chi'$ to $S \cup \{v\}$ and we color $v$ with $\tau(v)$, then the resulting $k$-coloring is proper.
- If we extend $\chi'$ to $S \cup \{v\}$ and we color $v$ with $\tau(v)$, then the absolute increase in the size of $V_{\tau(v)}$ is minimal (among all possible choices that satisfy the first constraint).

In other words, $v \notin V_{\tau(v)}$ and for every $i$, $1 \le i \le k$, if $v \notin V_i$, then $|\{w \notin V_i : (v, w) \in E\}| \ge |\{w \notin V_{\tau(v)} : (v, w) \in E\}|$. Since $v$ is not a conflict vertex for $\langle S, \chi \rangle$, such a color $\tau(v)$ always exists (but is possibly not unique). By the fact that $v$ is not a heavy vertex for $\langle S, \chi \rangle$, we know that $|\{w \notin V_{\tau(v)} : (v, w) \in E\}| < \epsilon n/3$. Therefore, if we remove from $G'$ for every $v \in Z$ all edges $(v, w) \in E$ with $w \notin V_{\tau(v)}$, then the resulting graph $G^*$ is obtained from $G$ by removal of less than $\epsilon n^2$ edges. It remains to show that the following coloring of $G^*$ is proper: We color each vertex $v \in S \cup X \cup Y$ with color $\chi'(v)$. Each vertex $v \in Z$ is colored with color $\tau(v)$. Now assume that this coloring is not proper. Then there must be an edge $(v, u)$ such that $v$ and $u$ are colored in the same color. Since $\chi$ is proper, all vertices in $X$ and $Y$ are isolated, and by the definition of $\tau(v)$ it is immediate that such an edge can only be between vertices in $Z$. But then we have that $u \notin V_{\tau(v)}$, and this implies that $(v, u)$ has been removed from $G'$. This yields a contradiction. $\square$

It remains to prove the feasibility preserving property.

LEMMA 8.5 (feasibility preserving property). *If for some $S \subseteq V$ every basis covered by $S$ is violated by certain $v \in S$, then the subgraph of $G$ induced by vertices in $S$ cannot be properly $k$-colored.*

*Proof.* The proof is by contradiction. Let us suppose there is a proper $k$-coloring $\chi$ of the subgraph of $G$ induced by the vertices in $S$. For every $U \subseteq S$, let $\chi_U$ denote the coloring $\chi$ restricted to vertex set $U$.

Let us observe that the set of bases covered by $S$ is not empty because it contains the "empty set" basis $(\emptyset, \chi_\emptyset)$. Therefore, there exists a basis (possibly one of many) $b = (U, \chi_U)$ covered by $S$ having the maximum size of set $U$. Since $b$ is violated by certain $v \in S$, either $v$ is a conflict vertex for $\langle U, \chi_U \rangle$ or $v$ is a heavy vertex for $\langle U, \chi_U \rangle$. Furthermore, since $\chi$ was assumed to be a proper $k$-coloring of $S$, $v$ cannot be a conflict vertex for $\langle U, \chi_U \rangle$ and therefore it must be a heavy vertex for $\langle U, \chi_U \rangle$. But this implies that $(U \cup \{v\}, \chi_{U \cup \{v\}})$ is a basis that is moreover covered by $S$. This yields a contradiction, because we assumed that there is no basis $(K, \chi_K)$ covered by $S$ having $|K| > |U|$. $\square$

Therefore, we summarize our discussion in this section with the following theorem that follows directly from Theorem 7.3 and Lemmas 8.4 and 8.5.

THEOREM 8.6. *There is a property tester for the graph $k$-coloring property with a query complexity of*

$$\mathcal{O}\left((k\,\epsilon^{-2}\,\ln(k/\epsilon^2))^2\right) = \widetilde{\mathcal{O}}(k^2/\epsilon^4)\ .$$

## 9. Hypergraph coloring.

We now want to extend the analysis from section 8 to obtain an efficient *property tester for hypergraph coloring*. A *hypergraph* is a pair $\mathcal{H} = (V, E)$ with a finite vertex set $V$ and the edge set $E \subseteq 2^V$. A hypergraph $\mathcal{H}$ is *$\ell$-uniform* if $|e| = \ell$ for all $e \in E$. (Notice that a 2-uniform hypergraph is a graph.) A *$k$-coloring* of a hypergraph $\mathcal{H}$ is an assignment $\chi : V \to \{1, \ldots, k\}$ of colors to the vertices of the hypergraph. A coloring is *proper* if no edge in $E$ is *monochromatic*, that is, if for every edge $e \in E$ there are $v, u \in e$ with $\chi(v) \neq \chi(u)$. If $\mathcal{H}$ has a proper $k$-coloring, then $\mathcal{H}$ is *$k$-colorable.* The $k$-coloring problem for hypergraphs consists of determining whether a given hypergraph is $k$-colorable.

We want to design a property tester for the $k$-coloring problem in $\ell$-uniform hypergraphs. An $\ell$-uniform hypergraph can be represented by a function $f : V^\ell \to \{0, 1\}$ that encodes its adjacency matrix. We use the standard distance measure (Definition 2.3) to measure the distance between hypergraphs. In terms of hypergraphs we can express this distance measure as follows.

DEFINITION 9.1. *An $\ell$-uniform hypergraph $\mathcal{H} = (V, E)$ is $\epsilon$-far from having a proper $k$-coloring if one has to remove more than $\epsilon n^\ell$ edges from $\mathcal{H}$ to obtain a hypergraph that has a proper $k$-coloring.*

Now, we discuss how to apply our framework to hypergraph coloring. Similarly to the graph coloring problem, we identify the ground set $\mathcal{C}$ with the set of vertices $V$ of the input hypergraph $\mathcal{H} = (V, E)$. Since in our representation $\mathcal{H}$ can be viewed as given by its adjacency matrix representation, we define the interpretation $I$ to map each set of vertices to the submatrix induced by these vertices. That is, for every $S \subseteq V$, we have $I(S) = S \times S \times \cdots \times S$. Clearly, the interpretation is $N^\ell$-bounded. Let $\langle S, \chi \rangle$ be a pair with $S \subseteq V$ and $\chi$ a proper $k$-coloring of vertices in $S$.

DEFINITION 9.2. *We say a vertex $v$ is* $i$-colorable *with respect to $\langle S, \chi \rangle$ if for every $e \in E$ with $v \in e$, either* (i) *there exists a vertex $u \in (S \cap e)$ with $\chi(u) \neq i$ or* (ii) *there exists a vertex $w \in e \setminus (S \cup \{v\})$.*

We want to extend our approach for graph coloring to hypergraphs. Again we use a recursive definition of bases that is based on *heavy* vertices and *conflict* vertices. In order to define heavy vertices we introduce a potential function. A vertex is a heavy vertex if its coloring increases the potential significantly, very much in the spirit of graph coloring; further motivation behind the definition of the potential function can be found in the proof of Lemma 9.7.

DEFINITION 9.3. *Let $\mathcal{H} = (V, E)$ be a hypergraph. Let $S \subseteq V$ and let $\chi$ be a proper $k$-coloring of vertices in $S$. The* potential *of $\langle S, \chi \rangle$ is defined as*

$$\Phi_{\mathcal{H}}(\langle S, \chi \rangle) \;=\; \sum_{i=1}^{k} \sum_{j=1}^{\ell-1} n^{j-1} \cdot |\varphi(\langle S, \chi \rangle, i, j)| \;\;,$$

*where*

$$\varphi(\langle S, \chi \rangle, i, j) \;=\; \big\{ W \subseteq V \;:\; |W| = \ell - j \;\;\&\;\; \exists e \in E \; (W \subseteq e \;\;\&\;\; \forall_{v \in e \setminus W} \;\; \chi(v) = i) \big\} \;.$$

Hence, if $W \in \varphi(\langle S, \chi \rangle, i, j)$, then coloring all vertices in $W$ with color $i$ creates a monochromatic edge. Our next step is to extend the notion of heavy and conflict vertices to hypergraphs.

DEFINITION 9.4. *A vertex $v \in V \setminus S$ is* heavy *with respect to $\langle S, \chi \rangle$ if* (i) *there is an $i$, $1 \le i \le k$, such that $v$ is $i$-colorable and* (ii) *for every $i$, $1 \le i \le k$, if $v$ is $i$-colorable and $\chi'$ is the extension of $\chi$ to $S \cup \{v\}$ by coloring $v$ with color $i$, then*

$$\Delta\Phi_{\mathcal{H}}(\langle S, \chi \rangle, v, i) \;=\; \Phi_{\mathcal{H}}(\langle S \cup \{v\}, \chi' \rangle) - \Phi_{\mathcal{H}}(\langle S, \chi \rangle) \;>\; \frac{\epsilon n^{\ell-1}}{3} \;\;.$$

DEFINITION 9.5. *A vertex $v \in V \setminus S$ is a* conflict vertex *with respect to $\langle S, \chi \rangle$ if for every $i$, $1 \le i \le k$, $v$ is* not *$i$-colorable.*

The bases and the violation function are defined in the same way as for graph coloring.

*Bases for $k$-coloring.*

- $\{\emptyset, 1\}$ is a basis (where 1 is the encoding of the coloring of the empty set of vertices).
- If $b = (K, \chi)$ is a basis, $v$ is a *heavy* vertex for $b$ and $\chi'$ is an encoding of the previous coloring $\chi$ of $K$ extended by a proper coloring of $v$, then $(K \cup \{v\}, \chi')$ is a basis.

*Violation function for $k$-coloring. A basis $b = (K, \chi)$ is* violated *by a vertex $v \in V$ if either* (i) *$v$ is a heavy vertex for $\langle K, \chi \rangle$ or* (ii) *$v$ is a conflict vertex for $\langle K, \chi \rangle$.*

In a manner similar to that for the graph coloring problem, we can give an upper bound for the dimension of the constructed ACP.

LEMMA 9.6. *For every problem instance of the hypergraph $k$-coloring problem the corresponding ACP $\mathcal{P}$ has* $\dim(\mathcal{P}) \le 3k\ell/\epsilon$ *and width$(\mathcal{P}) \le k^{3k\ell/\epsilon}$.*

*Proof.* Since every $k$-coloring of a set of $r$ vertices can be encoded using an integer in the range between 1 and $k^r$, it is enough to show that $|K| \le 3\,k\,\ell/\epsilon$ for every basis $b = (K, \chi)$. To show this, let us recall that the bases are defined inductively by adding a heavy vertex to another basis. By definition, a heavy vertex increases the potential of the corresponding basis by more than $\frac{1}{3}\,\epsilon\, n^{\ell-1}$. The maximum potential of every basis is less than $k\,\ell\, n^{\ell-1}$ and the starting potential is 0. Thus, it follows for every basis $b = (K, \chi)$ that $|K| \le 3\,k\,\ell/\epsilon$. $\square$

Our next step is to prove the distance preserving property.

LEMMA 9.7 (($\epsilon, \epsilon/3$)-distance preserving property). *Let $\mathcal{H} = (V, E)$ be a hypergraph that is $\epsilon$-far from being $k$-colorable, and let $S \subseteq V$ be an arbitrary set of vertices colored according to a proper $k$-coloring $\chi$. Then either $V$ contains more than $\epsilon\, n/3$ conflict vertices with respect to $\langle S, \chi \rangle$ or $V$ has more than $\epsilon\, n/3$ heavy vertices for $\langle S, \chi \rangle$.*

*Proof.* Our arguments are similar to those used in the proof of Lemma 8.4. The proof is by contradiction. Let us assume there are less than or equal to $\epsilon\, n/3$ heavy vertices and less than or equal to $\epsilon\, n/3$ conflict vertices with respect to $\langle S, \chi\rangle$. Then we show that it is possible to extend coloring $\chi$ of $S$ to a coloring $\chi^*$ of $V$ that has at most $\epsilon\, n^\ell$ monochromatic (violating) edges in $\mathcal{H}$. This would yield a contradiction.

We define $\chi^*$ as follows:

$$\chi^*(v) = \begin{cases} \chi(v) & \text{for every } v \in S;\\ 1 & \text{if } v \in V \setminus S \text{ and } v \text{ is a heavy vertex or a conflict vertex with} \\ & \quad \text{respect to } \langle S,\chi\rangle;\\ i & \text{if } v \in V\setminus S \text{ is } i\text{-colorable with respect to } \langle S,\chi\rangle \text{ and } i \text{ minimizes} \\ & \quad \text{(over all possible choices of proper coloring } i\text{) the increase in} \\ & \quad \text{the potential, that is, } \Delta\Phi_{\mathcal{H}}(\langle S,\chi\rangle, v, i) \le \Delta\Phi_{\mathcal{H}}(\langle S,\chi\rangle, v, j) \\ & \quad \text{for every proper coloring } j \text{ of } v. \end{cases}$$

Now, we give an upper bound on the number of monochromatic edges in coloring $\chi^*$ of $\mathcal{H}$. Let us first consider heavy and conflict vertices. By our assumption, the number of such vertices is upper bounded by $\frac{2}{3}\,\epsilon\, n$. Therefore, the number of edges incident to these vertices is upper bounded by $\frac{2}{3}\,\epsilon\, n^\ell$. Hence, it is sufficient to show that there are at most $\frac{1}{3}\,\epsilon\, n^\ell$ monochromatic edges in $\mathcal{H}$ that are not incident to heavy or conflict vertices. We show this indirectly by defining a set $E_D \subseteq E$ of at most $\frac{1}{3}\,\epsilon\, n^\ell$ edges. Then we prove that this set contains all monochromatic edges for the coloring $\chi^*$. Let $V_{light}$ denote the set of all vertices in $V \setminus S$ that are neither heavy nor conflict vertices for $\langle S, \chi\rangle$.

For a vertex $v \in V_{light}$ let us define

$$\Delta\varphi(\langle S,\chi\rangle, v, i, j) \;=\; \varphi(\langle S\cup\{v\}, \chi'\rangle, i, j) \setminus \varphi(\langle S,\chi\rangle, i, j) \enspace ,$$

where $\chi'$ is the extension of $\chi$ to $S \cup \{v\}$ by coloring vertex $v$ with color $i$. We further denote by $E(X, v) = \{e \in E : v \in e \;\&\; X \subseteq e\}$ all edges of the hypergraph that contain $X \cup \{v\}$. Now we make a simple but important observation.

Claim 9.8. *If $E(X,v) = \emptyset$, then $X \notin \Delta\varphi(\langle S,\chi\rangle, v, \chi^*(v), j)$.*

*Proof.* The proof follows immediately from the definition of $\Delta\varphi(\langle S,\chi\rangle, v, \chi^*(v), j)$. ☐

We define the set

$$E_D \;=\; \bigcup_{v\in V_{light}} \; \bigcup_{j\in[\ell-1]} E^{(v)}_{D,j}$$

using sets $E^{(v)}_{D,j}$ that determine for each vertex $v$ a set of edges that is responsible for the sets in $\Delta\varphi(\langle S,\chi\rangle, v, \chi^*(v), j)$. As we will see later, these edges are the only edges that may possibly be monochromatic in $\chi^*$. We define the set $E^{(v)}_{D,j}$ as follows:

$$E^{(v)}_{D,j} \;=\; \bigcup_{X\in\Delta\varphi(\langle S,\chi\rangle, v, \chi^*(v), j)} E(X, v) \enspace .$$

Claim 9.9. *Let $e \in E$ be a monochromatic edge in the coloring $\langle V, \chi^*\rangle$. Then $e \in E_D$.*

*Proof.* If $e$ is monochromatic, then there is $i \in [k]$ such that $\chi^*(u) = i$ for all $u \in e$. Furthermore, we conclude that for each $u \in e$ we have $\{u\} \in \varphi(\langle V,\chi\rangle, i, \ell - 1)$. Now we distinguish between two cases. First let us consider the case when there is a

$u \in e$ with $\{u\} \in \varphi(\langle S, \chi\rangle, i, \ell - 1)$. In this case $\chi^*(u) \neq i$ by the definition of $\chi^*$. In the second case such a $u$ does not exists, but we have $\{u\} \in \varphi(\langle V, \chi\rangle, i, \ell - 1)$. But as can be seen from Claim 9.8, we defined $E_D$ in such a way that there cannot be an $X \in \varphi(\langle V, \chi\rangle, i, \ell - 1)$ that is not contained in $\varphi(\langle S, \chi\rangle, i, \ell - 1)$. Hence, if $e$ is not in $E_D$, it cannot be monochromatic. ∎

It remains to show that the size of $E_D$ is small enough. By definition we have

$$\left|E_D\right| \; = \sum_{v \in V_{light}} \sum_{j=1}^{\ell-1} \left|E_{D,j}^{(v)}\right| \; ,$$

and it is easy to see that for the $E_{D,j}^{(v)}$ it holds that

$$\left|E_{D,j}^{(v)}\right| \; \leq \; \left|\Delta\varphi(\langle S, \chi\rangle, v, \chi^*(v), j)\right| \cdot n^{j-1}$$

because of the fact that $|E(X, v)| \leq n^{\ell-(|X|+1)}$. We conclude that we have

$$\left|E_D\right| \; \leq \sum_{v \in V_{light}} \sum_{j=1}^{\ell-1} \left|\Delta\varphi(\langle S, \chi\rangle, v, \chi^*(v), j)\right| \cdot n^{j-1} \; .$$

Since all considered vertices are light we also have, for every $v \in V_{light}$,

$$\Delta\Phi_{\mathcal{H}}(\langle S, \chi\rangle, v, \chi^*(v)) \; = \sum_{j=1}^{\ell-1} \left|\Delta\varphi(\langle S, \chi\rangle, v, \chi^*(v), j)\right| \cdot n^{j-1} \; \leq \; \frac{1}{3}\epsilon n^{\ell-1} \; .$$

This finally gives us

$$\left|E_D\right| \; \leq \; \frac{1}{3}\epsilon n^{\ell} \; .$$

Together with Claim 9.9 we get a contradiction to the fact that $\mathcal{H}$ is $\epsilon$-far from being $k$-colorable. This proves the lemma. ∎

The proof of the following lemma is essentially the same as the proof of Lemma 8.5, and we present it here for the sake of completeness only.

LEMMA 9.10 (feasibility preserving property). *If for some $S \subseteq V$ every basis covered by $S$ is violated by certain $v \in S$, then the subhypergraph of $\mathcal{H}$ induced by vertices in $S$ cannot be properly $k$-colored.*

*Proof.* The proof is by contradiction. Let us suppose there is a proper $k$-coloring $\chi$ of the subgraph of $\mathcal{H}$ induced by the vertices in $S$. For every $U \subseteq S$, let $\chi_{|U}$ denote the coloring $\chi$ restricted to vertex set $U$.

Let us observe that the set of bases covered by $S$ is not empty, because it contains the "empty set" basis $(\emptyset, \chi_{\emptyset})$. Therefore, there exists a basis (possibly one of many) $b = (U, \chi_{|U})$ covered by $S$ having the maximum size of set $U$. Since $b$ is violated by certain $v \in S$, either $v$ is a conflict vertex for $\langle U, \chi_{|U}\rangle$ or $v$ is a heavy vertex for $\langle U, \chi_{|U}\rangle$. Furthermore, since $\chi$ was assumed to be a proper $k$-coloring of $S$, $v$ cannot be a conflict vertex for $\langle U, \chi_U\rangle$, and therefore it must be a heavy vertex for $\langle U, \chi_{|U}\rangle$. But this implies that $(U \cup \{v\}, \chi_{|U\cup\{v\}})$ is a basis that is moreover covered by $S$. This yields a contradiction, because we assumed that there is no basis $(K, \chi_{|K})$ covered by $S$ having the size of $K$ greater than $|U|$. ∎

The three lemmas above combined with our framework from Theorem 7.3 imply the following result.

Theorem 9.11. *There is a property tester for the hypergraph $k$-colorability with the query complexity*

$$\mathcal{O}\left((k\,\ell\,\epsilon^{-2}\,\ln(k/\epsilon))^{\ell}\right) = \widetilde{\mathcal{O}}((k\,\ell/\epsilon^2)^{\ell}) \ .$$

**10. Testable hereditary graph properties.** In this section we consider arbitrary *hereditary graph properties.* A *graph property* $\Pi$ is a family of graphs that is preserved under graph isomorphism (that is, if $G$ satisfies property $\Pi$ and $G'$ is a graph isomorphic to $G$, then $G'$ has property $\Pi$, too). A graph property $\Pi$ is *hereditary* if it is closed under taking induced subgraphs; that is, if graph $G = (V, E)$ has $\Pi$, then every subgraph $G_S$ induced by a set $S \subseteq V$ has property $\Pi$ (see, e.g., [16]). Similarly as in section 8, we consider undirected graphs that are represented by a function $f : V \times V \to \{0, 1\}$ that encodes the adjacency matrix of the graph. We assume without loss of generality that $V = [n]$. When we talk about graph properties we have to observe that in our framework a property is defined as a subset of the set of $n$-vertex graphs. When no confusion can arise we use $\Pi$ to denote the graph property $\Pi$ as well as the corresponding set of $n$-vertex graphs (which is the formal definition of a property in this paper). Our main result is that a hereditary graph property is efficiently testable if and only if it can be reduced to an ACP of dimension that is independent of the size of the input graph.

For every graph $G = (V, E)$ and every subset $U \subseteq V$ we denote by $G_U$ the subgraph of $G$ induced by $U$.

We use the standard distance measure from Definition 2.3 for testing graph properties.

Definition 10.1. *Let* $\Pi$ *be an arbitrary graph property. A graph* $G$ *is* $\epsilon$-far from (satisfying) $\Pi$ *if in order to transform* $G$ *into a graph satisfying* $\Pi$ *one has to modify more than* $\epsilon\, n^2$ *entries in the adjacency matrix of* $G$.

Now, we can formally state the main result of this section.

Theorem 10.2. *Let* $\Pi$ *be a hereditary graph property. Let* $0 < \epsilon < 1$. *Let* $\mathcal{G}$ *be the set of all graphs on the vertex set* $V = [n]$. *Then* $\Pi$ *is efficiently testable* if and only if *there are* $\delta = \delta(\epsilon)$, $\varrho = \varrho(\epsilon)$, *and* $\lambda = \lambda(\epsilon)$, *such that every* $G \in \mathcal{G}$ *can be mapped to an ACP* $\mathcal{P}_G = ([n], \mathcal{B}, \varpi)$ *with* $\dim(\mathcal{P}_G) \le \delta(\epsilon)$ *and width*$(\mathcal{P}_G) \le \varrho(\epsilon)$ *satisfying the following two properties:*

- $((\epsilon, \lambda)$-distance preserving) *if* $G$ *is* $\epsilon$*-far from* $\Pi$, *then every basis in* $\mathcal{P}_G$ *is* $\lambda$*-far from feasible; and*
- (feasibility preserving) *for every* $S \subseteq V$ *with* $|S| \ge \delta(\epsilon)$*: If there is* $G' \in \Pi$ *with* $G_S = G'_S$, *then there is a self-feasible basis for* $S$ *in* $\mathcal{P}_G$.

The remaining part of this section is devoted to the proof of Theorem 10.2. We begin with the following lemma, which simplifies the analysis of property testers for graph properties. By this lemma, if a hereditary graph property $\Pi$ has a property tester with a query complexity of $q(\epsilon)$, then it has a property tester that samples a set $S$ of $r(q(\epsilon)) = \mathcal{O}(q(\epsilon))$ vertices and accepts if and only if the subgraph induced by $S$ has $\Pi$.

Lemma 10.3 (Alon; see [47, Proposition D.2]). *Let* $\Pi$ *be a hereditary graph property. Suppose there is a property tester for property* $\Pi$ *with query complexity* $q(\epsilon)$. *Then there is a property tester for property* $\Pi$ *that selects uniformly at random a set of* $r(q(\epsilon)) = \mathcal{O}(q(\epsilon))$ *vertices and accepts the input graph if and only if the corresponding induced subgraph satisfies property* $\Pi$.

In order to prove Theorem 10.2, we must prove that our condition is both necessary and sufficient for efficient testability of any property $\Pi$. We first observe that our proof of Theorem 7.3 with the interpretation $I$ as defined in section 8 implies directly the sufficiency of our conditions. Now, we prove the necessity of our condition (Lemma 10.4 uses the identical function $r()$ as in Lemma 10.3).

LEMMA 10.4 (necessary condition). *Let $\Pi$ be a hereditary graph property. Let $0 < \epsilon < 1$. Suppose there is a property tester for property $\Pi$ with query complexity $q(\epsilon) \le \frac{1}{4}n$, and let us set $r = r(q(\epsilon))$. Let $\mathcal{G}$ be the set of all graphs on the vertex set $V = \{1, \ldots, n\}$. Let $\lambda = \lambda(\epsilon) = \frac{1}{3 \cdot 2^r}$. Then for every $G \in \mathcal{G}$ there exists an ACP $\mathcal{P}_G = ([n], \mathcal{B}, \varpi)$ with $\dim(\mathcal{P}_G) \le 2r$ and width$(\mathcal{P}_G) = 1$ satisfying the $(\epsilon, \lambda)$-distance preserving and the feasibility preserving properties.*

*Proof.* Let us fix an arbitrary graph $G = (V, E)$ for which we describe ACP $\mathcal{P}_G$ satisfying the required properties.

*Bases for graph properties.* We define the bases of $\mathcal{P}_G$ to be of the form $(K, 1)$, where $K \subseteq V$. We first give a set of basis candidates $\mathcal{BC}$ and then show how to obtain the set of bases from this set of candidates.

We define the set of *basis candidates* $\mathcal{BC}$ as follows:
- $(\emptyset, 1)$ *is a basis candidate*;
- if $K$ is a set of vertices of size $r$ and $G_K$ (the subgraph induced by $K$) does not satisfies $\Pi$, then $(K, 1)$ *is a basis candidate*; and
- if $(K, 1)$ is a basis and $K' \subseteq K$, then $(K', 1)$ *is a basis candidate.*

We also define the notion of *violators* and *light* bases.

DEFINITION 10.5. *Let $\mathcal{BC}$ be a set of basis candidates and let $b = (K, 1) \in \mathcal{BC}$. The set of* violators *$Viol^{\mathcal{BC}}(b)$ of $b$ with respect to $\mathcal{BC}$ is defined as follows:*
- *if $|K| = r$, then $Viol^{\mathcal{BC}}(b) = V \setminus K$; and*
- *if $|K| < r$, then $Viol^{\mathcal{BC}}(b) = \{v \in V \setminus K :$ there exists a basis candidate $b' = (K \cup \{v\}, 1)\}$.*

DEFINITION 10.6. *Let $\mathcal{BC}$ be a set of basis candidates and let $b \in \mathcal{BC}$. We call $b$* light *(with respect to $\mathcal{BC}$) if $|Viol^{\mathcal{BC}}(b)| \le \frac{3}{2}\lambda n$. If $b$ is not light, then it is* heavy.

Now, we define the *set of bases* $\mathcal{B}$ as the output of the following algorithm COMPUTEBASES.

```
COMPUTEBASES (set of candidates $\mathcal{BC}$).
    while there is a light basis $b$ (with respect to the current $\mathcal{BC}$) do
        $\mathcal{BC} = \mathcal{BC} \setminus \{b\}$
    return $\mathcal{B} = \mathcal{BC} \cup \{(\emptyset, 1)\}$
```

*Violation function for graph properties. A basis $b \in \mathcal{B}$ is violated by its violators, that is, by all vertices contained in $Viol^{\mathcal{B}}(b)$.*

Notice that our construction of bases and the definition of the violation function implies that every basis $b \in \mathcal{B}$ is heavy; that is, it is violated by more than $\frac{3}{2}\lambda n$ vertices in $V$.

Now, the following claim follows trivially from our construction.

CLAIM 10.7. *$\mathcal{P}_G$ has dimension $(r, 1)$.*

*$(\epsilon, \lambda)$-distance preserving property.* We prove now that the ACP $\mathcal{P}_G$ defined above satisfies the $(\epsilon, \lambda)$-distance preserving property; that is, we prove that if $G$ is $\epsilon$-far from $\Pi$, then every basis in $\mathcal{P}_G$ is $\lambda$-far from feasible, where $\lambda = \lambda(\epsilon) = \frac{1}{3 \cdot 2^r}$. Notice that by the definition of bases, every basis except for $(\emptyset, 1)$ is violated by more than $\lambda n$ ground set elements. Therefore, what remains to be proved is that the basis $(\emptyset, 1)$ is also violated by more than $\lambda n$ ground set elements. Our proof of this fact is via a sequence of claims that top-down establish lower bounds for the number of bases of given size.

For a basis $b = (K, 1)$, let $|K|$ be called its *size.* We begin with the following simple claim about bases of size $r$.

CLAIM 10.8. *Suppose $r < (1 - \frac{2}{3}\lambda)\, n$. If $G$ is $\epsilon$-far from $\Pi$, then the number of bases of size $r$ in $\mathcal{B}$ is bigger than $\frac{2}{3} \cdot \binom{n}{r}$.*

*Proof.* The proof follows directly from the definition of the bases and from Lemma 10.3. Indeed, Lemma 10.3 implies that for a graph that is $\epsilon$-far from $\Pi$, if one picks at random a set of $r$ vertices in $V$, then with probability at least $\frac{2}{3}$ the subgraph induced by these vertices does not satisfy $\Pi$. This is equivalent to saying that the number of subsets of $V$ of size $r$ for which the induced subgraph does not satisfy $\Pi$ is bigger than $\frac{2}{3} \cdot \binom{n}{r}$. By the definition of the basis candidates, for every set $K \subseteq V$ of size $r$, $(K, 1) \in \mathcal{BC}$. Moreover, the set $V \setminus K$ is the set of violators of $K$. Hence $(K, 1)$ is a basis that is violated by $n - r$ violators, and thus $(K, 1)$ is heavy. Thus, $(K, 1)$ belongs to $\mathcal{B}$. Therefore, the number of bases of size $r$ in $\mathcal{B}$ is bigger than $\frac{2}{3} \cdot \binom{n}{r}$. □

The next claim deals with the relation between the number of bases of size $k$ and of size $k - 1$.

CLAIM 10.9. *Suppose that $n \geq 2\,r$. Let $\zeta$, $0 \leq \zeta \leq 1$, and let $k$, $1 \leq k \leq r$, be an integer. If there are more than $\zeta \cdot \binom{n}{k}$ bases of size $k$ in $\mathcal{B}$, then the number of bases in $\mathcal{B}$ of size $k - 1$ is bigger than $\frac{\zeta - 3 \cdot \lambda}{2} \cdot \binom{n}{k-1}$.*

*Proof.* Recall that $\mathcal{B}$ contains only heavy bases. Furthermore, if $(K, 1)$ is a basis, then our construction of bases ensures that for every $u \in K$, $(K \setminus \{u\}, 1)$ is a basis (in the following, "basis" also refers to basis candidates) and that this basis is violated by $u$. Thus, every basis of size $k$ defines $k$ violators for bases for size $k - 1$. We conclude that overall, there are more than

$$k \cdot \zeta \cdot \binom{n}{k} = (n - k + 1) \cdot \zeta \cdot \binom{n}{k-1} > \frac{1}{2} \cdot n \cdot \zeta \cdot \binom{n}{k-1}$$

violators for bases of size $k - 1$. Observe that every light basis has at most $\frac{3}{2}\,\lambda\, n$ violators. Therefore, the number of violators of all light bases of size $k - 1$ is at most $\frac{3}{2}\,\lambda\, n\, \binom{n}{k-1}$. It follows that the number of violators of heavy bases of that size is bigger than

$$\frac{1}{2} \cdot n \cdot \zeta \cdot \binom{n}{k-1} - \frac{3}{2} \cdot \lambda \cdot n \cdot \binom{n}{k-1} = \frac{1}{2} \cdot (\zeta - 3\,\lambda) \cdot n \cdot \binom{n}{k-1} \ .$$

Since each basis has at most $n$ violators it follows that the number of heavy bases is larger than

$$\frac{1}{2} \cdot (\zeta - 3\,\lambda) \cdot \binom{n}{k-1} \ ,$$

which completes the proof of our claim. □

The next claim gives a lower bound for the number of bases of given size (bigger than 0).

CLAIM 10.10. *Let $1 \leq k \leq r$ be an integer. Then the number of bases of size $r - k$ is bigger than*

$$\left( \frac{1}{3 \cdot 2^{k-1}} - \frac{3}{2}\,\lambda \left( 2 - \frac{1}{2^{k-1}} \right) \right) \cdot \binom{n}{r-k} \ .$$

*Proof.* The proof is by induction on $k$. For $k = 1$ the claim follows directly from Claims 10.8 and 10.9. Now, let us assume the claim is true for $k = k'$, for certain

$1 \le k' < r$, and we show that this implies that the claim is true also for $k = k' + 1$. By induction, this will yield the claim.

By the induction hypothesis, the number of bases in $\mathcal{B}$ of size $r - k'$ is bigger than

$$\begin{aligned}&\left(\frac{1}{3 \cdot 2^{k'-1}} - \frac{3}{2}\,\lambda\,(2 - \frac{1}{2^{k'-1}})\right) \cdot \binom{n}{r-k'}\\ &= \left(\frac{1}{3 \cdot 2^{k-2}} - \frac{3}{2}\,\lambda\left(2 - \frac{1}{2^{k-2}}\right)\right) \cdot \binom{n}{r-k+1}\ .\end{aligned}$$

Therefore, by Claim 10.9, the number of bases in $\mathcal{B}$ of size $r - k = (r - k') - 1$ is bigger than

$$\frac{\left(\frac{1}{3 \cdot 2^{k-2}} - \frac{3}{2}\lambda\left(2 - \frac{1}{2^{k-2}}\right)\right) - 3\,\lambda}{2} \cdot \binom{n}{r-k} = \left(\frac{1}{3 \cdot 2^{k-1}} - \frac{3}{2}\,\lambda\left(2 - \frac{1}{2^{k-1}}\right)\right) \cdot \binom{n}{r-k},$$

and the claim follows. $\square$

Now, we are ready to complete the proof of the $(\epsilon, \lambda)$-distance preserving property for $\mathcal{P}_G$. By our discussion above, we must show only that the basis $(\emptyset, 1)$ is violated by more than $\lambda\, n$ ground set elements. By Claim 10.10, the number of bases in $\mathcal{B}$ of size 1 is bigger than

$$\left(\frac{1}{3 \cdot 2^{r-2}} - 3 \cdot \lambda\right) \cdot \binom{n}{1} \;=\; \left(\frac{4}{3 \cdot 2^{r}} - 3 \cdot \lambda\right) \cdot n \;\ge\; \lambda \cdot n$$

by our assumption that $\lambda = \frac{1}{3 \cdot 2^r}$. Each of the bases of size 1 is of the form $(\{u\}, 1)$ for some $u \in V$, and by the definition of violation, such a vertex $u$ violates $(\emptyset, 1)$. Since the number of such vertices $u$ is bigger than $\lambda \cdot n$, this completes the proof of the $(\epsilon, \lambda)$-distance preserving property for $\mathcal{P}_G$.

*Feasibility preserving property.* We prove now that the ACP $\mathcal{P}_G$ satisfies the feasibility preserving property. We prove that for every $S \subseteq V$, $|S| \ge 2\,r$, if the subgraph $G_S$ satisfies $\Pi$, then there is a self-feasible basis for $S$ in $\mathcal{P}_G$. This implies the feasibility preserving property because $\Pi$ is hereditary, and so if $G_S$ does not have property $\Pi$, then $G$ cannot have $\Pi$, as well.

Our arguments are roughly the same as those in the proof of Lemma 8.5. The proof is by contradiction. Let us suppose that the subgraph $G_S$ satisfies $\Pi$, but every basis in $S$ is violated. Observe that the set of bases covered by $S$ is not empty, because it contains the "empty set" basis $(\emptyset, 1)$. Therefore, there exists a basis $b = (K, 1)$ covered by $S$ maximizing the size of set $K$. Since $b$ is violated by certain $v \in S$ and since $K$ is of maximum size, we conclude that $|K| = r$. It follows that the subgraph $G_K$ induced by $K$ does not have property $\Pi$. Since $\Pi$ is hereditary, it is closed under taking induced subgraphs, and hence we conclude that $G_S$ does not have property $\Pi$.

If $n < 2r$, then we use the following simple ACP: The ACP has a single basis $(V, 1)$ which is violated by all ground set elements if $G$ is $\epsilon$-far from $\Pi$, and which is not violated otherwise. This completes the proof of Lemma 10.4. $\square$

Now, we can conclude our discussion to complete the proof of Theorem 10.2.

*Proof.* The "only if" part follows from Lemma 10.4, and the "if" part follows from Theorem 7.3 with the interpretation $I$ as defined in section 8. $\square$

## 11. Conclusions.

In this paper we introduced a novel framework that provides a sufficient condition for property testing in the one-sided error model. We demonstrated how to apply our framework on several problems from different areas, including computational geometry, graph theory, and computational biology. For some of these problems we presented the first property tester. For others, we improved existing bounds on the query complexity of property testers for these problems. We also proved that for hereditary graph properties our framework provides a sufficient and necessary condition for testability: we showed that any hereditary graph property is testable with query complexity independent of the input graph if and only if this can be proven using our framework.

Perhaps one of the most interesting open problems left in the paper concerns extending our framework to the two-sided error model. Another exciting problem is to design a similar framework for the sparse graph model. Compared to our framework, one difficulty with testing properties of sparse graphs is the fact that property testers use adaptive sampling techniques, e.g., random walks or graph traversals starting from random vertices, rather than uniform sampling. Consequently, one would have to find a randomized process that covers most of these adaptive sampling techniques.

**Acknowledgments.** We would like to thank Emo Welzl and an anonymous referee for their helpful comments on this work.

## REFERENCES

[1] P. K. Agarwal and C. M. Procopiuc, *Exact and approximation algorithms for clustering*, Algorithmica, 33 (2002), pp. 201–226.

[2] N. Alon, *Testing subgraphs in large graphs*, Random Structures Algorithms, 21 (2002), pp. 359–370.

[3] N. Alon, S. Dar, M. Parnas, and D. Ron, *Testing of clustering*, SIAM J. Discrete Math., 16 (2003), pp. 393–417.

[4] N. Alon, E. Fischer, M. Krivelevich, and M. Szegedy, *Efficient testing of large graphs*, Combinatorica, 20 (2000), pp. 451–476.

[5] N. Alon, W. Fernandez de la Vega, R. Kannan, and M. Karpinski, *Random sampling and approximation of MAX-CSP problems*, J. Comput. System Sci., 67 (2003), pp. 212–243.

[6] N. Alon, M. Krivelevich, I. Newman, and M. Szegedy, *Regular languages are testable with a constant number of queries*, SIAM J. Comput., 30 (2001), pp. 1842–1862.

[7] N. Alon and M. Krivelevich, *Testing k-colorability*, SIAM J. Discrete Math., 15 (2002), pp. 211–227.

[8] N. Alon and A. Shapira, *Testing satisfiability*, J. Algorithms, 47 (2003), pp. 87–103.

[9] S. Arora, C. Lund, R. Motwani, M. Sudan, and M. Szegedy, *Proof verification and hardness of approximation problems*, J. ACM, 45 (1998), pp. 501–555.

[10] G. Ausiello, P. Crescenzi, G. Gambosi, V. Kann, A. Marchetti-Spaccamela, and M. Protasi, *Complexity and Approximation. Combinatorial Optimization Problems and Their Approximability Properties*, Springer-Verlag, New York, 1998.

[11] M. Badoiu, S. Har-Peled, and P. Indyk, *Approximate clustering via core-sets*, in Proceedings of the 34th Annual ACM Symposium on Theory of Computing (STOC), ACM Press, New York, 2002, pp. 250–257.

[12] V. Bafna and P. A. Pevzner, *Genome rearrangements and sorting by reversals*, SIAM J. Comput., 25 (1996), pp. 272–289.

[13] P. Berman, S. Hannenhalli, and M. Karpinski, 1.375-*approximation algorithm for sorting by reversals*, in Proceedings of the 10th Annual European Symposium on Algorithms (ESA), Lecture Notes in Comput. Sci. 2461, R. H. Möhring and R. Raman, eds., Springer-Verlag, Berlin, 2002, pp. 200–210.

[14] P. Berman and M. Karpinski, *On some tighter inapproximability results, further improvements*, in Proceedings of the 26th Annual International Colloquium on Automata, Languages and Programming (ICALP), Lecture Notes in Comput. Sci. 1644, J. Wiedermann, P. van Emde Boas, and M. Nielsen, eds., Springer-Verlag, Berlin, 1999, pp. 200–209.

[15] M. Blum, M. Luby, and R. Rubinfeld, *Self-testing/correcting with applications to numerical problems*, J. Comput. System Sci., 47 (1993), pp. 549–595.

[16] B. Bollobás, *Hereditary properties of graphs: Asymptotic enumeration, global structure, and colouring*, in Proceedings of the International Congress of Mathematicians, Vol. III, 1998, pp. 333–342.

[17] H. Buhrman, L. Fortnow, I. Newman, and H. Röhrig, *Quantum property testing*, in Proceedings of the 14th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA), SIAM, Philadelphia, 2003, pp. 480–488.

[18] A. Caprara, *Sorting permutations by reversals and Eulerian cycle decompositions*, SIAM J. Discrete Math., 12 (1999), pp. 91–110.

[19] B. Chazelle, R. Rubinfeld, and L. Trevisan, *Approximating the minimum spanning tree weight in sublinear time*, in Proceedings of the 28th Annual International Colloquium on Automata, Languages and Programming (ICALP), Lecture Notes in Comput. Sci. 2076, F. Orejas, P. G. Spirakis, and J. van Leeuwen, eds., Springer-Verlag, Berlin, 2001, pp. 190–200.

[20] A. Czumaj, F. Ergün, L. Fortnow, A. Magen, I. Newman, R. Rubinfeld, and C. Sohler, *Sublinear-time approximation of Euclidean minimum spanning tree*, in Proceedings of the 14th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA), SIAM, Philadelphia, 2003, pp. 813–822.

[21] A. Czumaj and C. Scheideler, *An algorithmic approach to the general Lovász Local Lemma with applications to scheduling and satisfiability problems*, in Proceedings of the 32nd Annual ACM Symposium on Theory of Computing (STOC), ACM Press, New York, 2000, pp. 38–47.

[22] A. Czumaj and C. Scheideler, *Coloring non-uniform hypergraphs: A new algorithmic approach to the general Lovász Local Lemma*, Random Structures Algorithms, 17 (2000), pp. 213–237.

[23] A. Czumaj and C. Sohler, *Testing hypergraph coloring*, in Proceedings of the 28th Annual International Colloquium on Automata, Languages and Programming (ICALP), Lecture Notes in Comput. Sci. 2076, F. Orejas, P. G. Spirakis, and J. van Leeuwen, eds., Springer-Verlag, Berlin, 2001, pp. 493–505.

[24] A. Czumaj and C. Sohler, *Property testing with geometric queries*, in Proceedings of the 9th Annual European Symposium on Algorithms (ESA), Lecture Notes in Comput. Sci. 2161, Springer-Verlag, Berlin, 2001, pp. 266–277.

[25] A. Czumaj and C. Sohler, *Sublinear-time approximation for clustering via random sampling*, in Proceedings of the 31st Annual International Colloquium on Automata, Languages and Programming (ICALP), Lecture Notes in Comput. Sci. 3142, J. Diaz, J. Karhumaki, A. Lepisto, and D. Sannella, eds., Springer-Verlag, Berlin, 2004, pp. 396–407.

[26] A. Czumaj and C. Sohler, *Estimating the weight of metric minimum spanning trees in sublinear-time*, in Proceedings of the 36th Annual ACM Symposium on Theory of Computing (STOC), ACM Press, New York, 2004, pp. 175–183.

[27] A. Czumaj, C. Sohler, and M. Ziegler, *Property testing in computational geometry*, in Proceedings of the 8th Annual European Symposium on Algorithms (ESA), Lecture Notes in Comput. Sci. 1879, M. Paterson, ed., Springer-Verlag, Berlin, 2000, pp. 155–166.

[28] P. Drineas, A. Frieze, R. Kannan, S. Vempala, and V. Vinay, *Clustering in large graphs and matrices*, in Proceedings of the 10th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA), SIAM, Philadelphia, 1999, pp. 291–299.

[29] Y. Dodis, O. Goldreich, E. Lehman, S. Raskhodnikova, D. Ron, and A. Samorodnitsky, *Improved testing algorithms for monotonicity*, in Proceedings of the 3rd International Workshop on Randomization and Approximation Techniques in Computer Science, Lecture Notes in Comput. Sci. 1671, D. Hochbaum, K. Jansen, J. D. P. Rolim, and A. Sinclair, eds., Springer-Verlag, Berlin, 1999, pp. 97–108.

[30] P. Erdős and L. Lovász, *Problems and results on 3-chromatic hypergraphs and some related questions*, in Infinite and Finite Sets (to Paul Erdős on his 60th birthday), A. Hajnal, R. Rado, and V. T. Sós, eds., Vol. II, North-Holland, Amsterdam, 1975, pp. 609–627.

[31] L. Engebretsen and J. Holmerin, *Towards optimal lower bounds for clique and chromatic number*, Theoret. Comput. Sci., 299 (2003), pp. 537–584.

[32] T. Feder and D. H. Greene, *Optimal algorithms for approximate clustering*, in Proceedings of the 20th Annual ACM Symposium on Theory of Computing (STOC), ACM Press, New York, 1998, pp. 434–444.

[33] W. Fernandez de la Vega and C. Kenyon, *A randomized approximation scheme for metric Max-Cut*, J. Comput. System Sci., 63 (2001), pp. 531–541.

[34] E. Fischer, *The art of uninformed decisions. A primer to property testing*, Bull. Eur. Assoc. Theor. Comput. Sci. EATCS, 75 (2001), pp. 97–126.

[35] E. Fischer, *Testing graphs for colorability properties*, in Proceedings of the 12th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA), SIAM, Philadelphia, 2001, pp. 873–882.

[36] E. Fischer, E. Lehman, I. Newman, S. Raskhodnikova, R. Rubinfeld, and A. Samorod-

NITSKY, *Monotonicity testing over general poset domains*, in Proceedings of the 34th Annual ACM Symposium on Theory of Computing (STOC), ACM Press, New York, 2002, pp. 474–483.

[37] E. FISCHER AND I. NEWMAN, *Testing of matrix properties*, in Proceedings of the 33rd Annual ACM Symposium on Theory of Computing (STOC), ACM Press, New York, 2001, pp. 286–295.

[38] E. FISCHER AND I. NEWMAN, *Functions that have read-twice constant width branching programs are not necessarily testable*, in Proceedings of the 17th Conference on Computational Complexity, IEEE Computer Society Press, Los Alamitos, CA, 2002, pp. 73–79.

[39] A. FRIEZE AND R. KANNAN, *The regularity lemma and approximation schemes for dense problems*, in Proceedings of the 37th IEEE Symposium on Foundations of Computer Science (FOCS), IEEE Computer Society Press, Los Alamitos, CA, 1996, pp. 12–20.

[40] A. FRIEZE, R. KANNAN, AND S. VEMPALA, *Fast Monte-Carlo algorithms for finding low-rank approximations*, in Proceedings of the 39th IEEE Symposium on Foundations of Computer Science (FOCS), IEEE Computer Society Press, Los Alamitos, CA, 1998, pp. 370–378.

[41] M. R. GAREY AND D. S. JOHNSON, *Computers and Intractability: A Guide to the Theory of NP-completeness*, Freeman, New York, 1979.

[42] O. GOLDREICH, *Combinatorial property testing (a survey)*, in Proceedings of the DIMACS Workshop on Randomization Methods in Algorithm Design, 1997, DIMACS Ser. Discrete Math. Theoret. Comput. Sci. 43, AMS, Providence, RI, 1999, pp. 45–59.

[43] O. GOLDREICH, S. GOLDWASSER, AND D. RON, *Property testing and its connection to learning and approximation*, J. ACM, 45 (1998), pp. 653–750.

[44] O. GOLDREICH, S. GOLDWASSER, E. LEHMAN, D. RON, AND A. SAMORODNITSKY, *Testing monotonicity*, Combinatorica, 20 (2000), pp. 301–337.

[45] O. GOLDREICH AND D. RON, *Property testing in bounded degree graphs*, Algorithmica, 32 (2002), pp. 302–343.

[46] O. GOLDREICH AND D. RON, *A sublinear bipartiteness tester for bounded degree graphs*, Combinatorica, 19 (1999), pp. 335–373.

[47] O. GOLDREICH AND L. TREVISAN, *Three theorems regarding testing graph properties*, Random Structures Algorithms, 23 (2003), pp. 23–57.

[48] V. GURUSWAMI, J. HÅSTAD, AND M. SUDAN, *Hardness of approximate hypergraph coloring*, SIAM J. Comput., 31 (2002), pp. 1663–1686.

[49] S. HAR-PELED, *Clustering motion*, in Proceedings of the 42nd IEEE Symposium on Foundations of Computer Science (FOCS), IEEE Computer Society Press, Los Alamitos, CA, 2001, pp. 84–93.

[50] D. HAUSSLER AND E. WELZL, *Epsilon-nets and simplex range queries*, Discrete Comput. Geom., 2 (1987), pp. 127–151.

[51] D. S. HOCHBAUM, ED., *Approximation Algorithms for $\mathcal{NP}$-Hard Problems*, PWS, Boston, MA, 1996.

[52] P. INDYK, *Sublinear time algorithms for metric space problems*, in Proceedings of the 30th Annual ACM Symposium on Theory of Computing (STOC), ACM Press, New York, 1998, pp. 428–434.

[53] P. INDYK, *A sublinear time approximation scheme for clustering in metric spaces*, in Proceedings of the 39th IEEE Symposium on Foundations of Computer Science (FOCS), IEEE Computer Society Press, Los Alamitos, CA, 1998, pp. 154–159.

[54] J. D. KECECIOGLU AND D. SANKOFF, *Exact and approximation algorithms for sorting by reversals, with application to genome rearrangement*, Algorithmica, 13 (1995), pp. 180–210.

[55] S. KHOT, *Hardness results for approximate hypergraph coloring*, in Proceedings of the 34th Annual ACM Symposium on Theory of Computing (STOC), ACM Press, New York, 2002, pp. 351–359.

[56] Y. KOHAYAKAWA, B. NAGLE, AND V. RÖDL, *Efficient testing of hypergraphs*, in Proceedings of the 29th Annual International Colloquium on Automata, Languages and Programming (ICALP), Lecture Notes in Comput. Sci. 2380, P. Widmayer, F. T. Ruiz, R. Morales, M. Hennessy, S. Eidenbenz, and R. Conejo, eds., Springer-Verlag, Berlin, 2002, pp. 1017–1028.

[57] M. KRIVELEVICH AND B. SUDAKOV, *Approximate coloring of uniform hypergraphs*, J. Algorithms, 49 (2003), pp. 2–12.

[58] M. LEWIS AND M. YANNAKAKIS, *The node deletion problem for hereditary properties is $\mathcal{NP}$-complete*, J. Comput. System Sci., 20 (1980), pp. 219–230.

[59] L. LOVÁSZ, *Coverings and colorings of hypergraphs*, in Proceedings of the 4th Southeastern Conference on Combinatorics, Graph Theory, and Computing, Utilitas Mathematica, Winnipeg, 1973, pp. 3–12.

[60] J. Matoušek, M. Sharir, and E. Welzl, *A subexponential bound for linear programming*, Algorithmica, 16 (1996), pp. 498–516.

[61] I. Newman, *Testing membership in languages that have small width branching programs*, SIAM J. Comput., 31 (2002), pp. 1557–1570.

[62] M. Parnas and D. Ron, *Testing metric properties*, in Proceedings of the 33rd Annual ACM Symposium on Theory of Computing (STOC), ACM Press, New York, 2001, pp. 276–285.

[63] P. A. Pevzner, *Computational Molecular Biology*, MIT Press, Cambridge, MA, 2000.

[64] P. A. Pevzner and M. S. Waterman, *Open combinatorial problems in computational molecular biology*, in Proceedings of the 3rd Israel Symposium on Theory of Computing and Systems (ISTCS), IEEE Computer Society Press, Los Alamitos, CA, 1995, pp. 158–173.

[65] G. Pisier, The Volume of Convex Bodies and Banach Space Geometry, Cambridge University Press, Cambridge, UK, 1989.

[66] J. Radhakrishnan and A. Srinivasan, *Improved bounds and algorithms for hypergraph two-coloring*, Random Structures Algorithms, 16 (2000), pp. 4–32.

[67] R. L. Rivest and J. Vuillemin, *On recognizing graph properties from adjacency matrices*, Theoret. Comput. Sci., 3 (1976), pp. 371–384.

[68] D. Ron, *Property testing*, in Handbook of Randomized Algorithms, Vol. II, P. M. Pardalos, S. Rajasekaran, J. Reif, and J. D. P. Rolim, eds., Kluwer Academic, Dordrecht, The Netherlands, 2001, pp. 597–649.

[69] R. Rubinfeld and M. Sudan, *Robust characterizations of polynomials with applications to program testing*, SIAM J. Comput., 25 (1996), pp. 252–271.

[70] M. Sudan, *Efficient Checking of Polynomials and Proofs and the Hardness of Approximation Problems*, Lecture Notes in Comput. Sci. 1001, Springer-Verlag, Heidelberg, 1996.

[71] V. N. Vapnik and A. Y. Chervonenkis, *On the uniform convergence of relative frequencies of events to their probabilities*, Theory Probab. Appl., 16 (1971), pp. 264–280.

[72] E. Welzl, *Smallest enclosing disks (balls and ellipsoids)*, in New Results and New Trends in Computer Science, Lecture Notes in Comput. Sci. 555, H. Maurer, ed., Springer-Verlag, Heidelberg, 1991, pp. 359–370.

SIAM J. COMPUT.
Vol. 34, No. 3, pp. 616–639

# LOAD BALANCING IN ARBITRARY NETWORK TOPOLOGIES WITH STOCHASTIC ADVERSARIAL INPUT[*]

ARIS ANAGNOSTOPOULOS[†], ADAM KIRSCH[‡], AND ELI UPFAL[†]

**Abstract.** We study the long-term (steady state) performance of a simple, randomized, local load balancing technique under a broad range of input conditions. We assume a system of $n$ processors connected by an arbitrary network topology. Jobs are placed in the processors by a deterministic or randomized adversary. The adversary knows the current and past load distribution in the network and can use this information to place the new tasks in the processors. A node can execute one job per step, and can also participate in one load balancing operation in which it can move tasks to a direct neighbor in the network. In the protocol we analyze here, a node equalizes its load with a random neighbor in the graph.

Our analysis of the protocol does not assume any particular input distribution. The input is generated by an arbitrary deterministic or probabilistic adversary subject only to some weak statistical properties. For stability and expected performance of the system we adopt the stochastic adversary model of [Borodin et al., *J. ACM*, 48 (2001), pp. 13–38]. For high-probability bounds we introduce a more restricted input model, the *strongly bounded* adversary.

Assuming the stochastic adversarial input model, we show that if the adversary does not trivially overload the network (i.e., there is an integer $w \geq 1$ such that the expected number of new jobs in any interval of length $w$ is bounded by $\lambda n w$ for some $\lambda < 1$), then the system is stable for any connected network topology, regardless of how the adversary allocates the new jobs between the processors.

When the system is stable, the next performance parameter of interest is the waiting time of jobs. We develop expected and high probability bounds on the total load in the system and the waiting time of jobs in terms of the network topology. In particular, in the above stochastic adversary model, if the network is an expander graph, the expected wait of a task is $O(w + \log n)$, and in the strongly bounded adversary model the waiting time of a task is $O(w + \log n)$ with high probability.

We contrast these results with the work stealing load balancing protocol, where we show that in sparse networks, the load in the system and the waiting time can be exponential in the network size.

**Key words.** dynamic load balancing, stochastic adversary, stability, efficiency, steady state analysis

**AMS subject classifications.** 68W40, 68W20, 60G35

**DOI.** 10.1137/S0097539703437831

## 1. Introduction.

Efficient utilization of parallel and distributed systems can often depend on dynamic load balancing of individual tasks between processors. In the dynamic load balancing problem, we consider a system that is designed to run indefinitely. New jobs arrive during the run of the system, and existing jobs are executed by the processors and leave the system. The arrival of new jobs may not be evenly distributed between the processors. The task of the load balancing protocol is to maintain approximately uniform job load between the processors, and in particular to keep all processors working as long as there are jobs in the system waiting for execution.

We assume a simple combinatorial model of load balancing following a number of earlier studies. The computing system is represented by a connected, undirected,

[*]Received by the editors November 20, 2003; accepted for publication (in revised form) August 20, 2004; published electronically March 17, 2005.
http://www.siam.org/journals/sicomp/34-3/43783.html

[†]Department of Computer Science, Brown University, Providence, RI 02912 (aris@cs.brown.edu, eli@cs.brown.edu).

[‡]Division of Engineering and Applied Sciences, Harvard University, Cambridge, MA 02138 (kirsch @eecs.harvard.edu). Part of this work was done while the author was at Brown University, and was supported in part by NSF grants CCR-0121154 and DMI-0121495.

$n$-node graph. Jobs (represented by tokens) have equal execution time. The load of a node is the number of tokens in its queue. A processor can execute (or consume) one token per step, and we assume that it executes the oldest job in its queue. In each step a processor can also move a number of jobs from its queue to the queue of an adjacent node in the network. This abstraction models the case where the execution time of a job is significantly longer than the time required to move a job to an adjacent node. The assumption that all jobs have equal execution time simplifies the analysis while still capturing the combinatorial complexity of the load balancing problem in networks.

Dynamic load balancing algorithms have been studied extensively in experimental settings, demonstrating significant run-time improvements obtained by relatively simple load balancing techniques [20, 21]. Rigorous, theoretical study of load balancing in the past has focused mainly on static analysis [1, 7, 8, 11, 16, 18], where a set of jobs is initially placed in the processors and the algorithm needs to distribute the jobs almost evenly between the processors in a minimum number of parallel rounds. A number of important techniques have been developed in this line of work, and in particular our work here builds on the static analysis in [8].

Load balancing, however, is best analyzed in a dynamic setting that captures the actual application of such protocols. An important step in that direction was taken in [4], where the work stealing model was shown to be stable on the complete network. The main tool used in that work was the stability conditions for ergodic Markov chains, and consequently their stability result holds only for Markovian adversaries. In addition, a number of other works have studied dynamic load balancing under the assumptions that jobs are generated by a randomized process that is oblivious to the current state of the system [10, 13, 14, 19]. Finally, [3, 15] proved stability results for load balancing on a general network assuming the deterministic adversarial model. The computational model there is different, assuming that only one job can traverse an edge per step.

## 2. Model and main results.

In this work we address both the stability and the efficiency (waiting time) of the load balancing task. We present a simple local randomized protocol and analyze its performance on a general $n$-node network, and under several adversarial models for the arrival of jobs into the system. Since we do not use Markov chain techniques in the analysis, our results are not restricted to Markovian adversaries. That is, the process that injects new jobs into the system can use information about all previous steps.

In particular, we consider the type of adversaries proposed by Borodin et al. in [5]. Following that work we define a $(w, \lambda n)$ input adversary as a process that inserts jobs in the system subject to the condition that for every sequence of $w$ consecutive time steps, the total inserted load is at most $\lambda n w$. This allows the adversary to insert more jobs at some time steps, as long as the total load in windows of size $w$ is bounded. An extension is a $(w, \lambda n)$ stochastic adversary, whose input load is a random variable, with the property that the expected injected load during any sequence of $w$ consecutive time steps is bounded by $\lambda n w$, and additionally, for some $p > 2$, the $p$th moment of the new load is bounded (see [5] for a detailed discussion).

For these adversaries, we derive asymptotic (with respect to the network size) bounds on both the expected total load and the waiting time of a job in the system. However, it is still reasonable to seek stronger guarantees than bounds on expected performance can provide. In particular, one might wish to augment the expected performance results with high-probability bounds. In this case, the stochastic adversarial model is too powerful, since it allows for large bursts of load to occur in certain time steps with decent probability. Therefore we need to place additional restrictions on the adversary if we wish to derive high-probability results. To this end, we introduce a constrained version of the stochastic adversary by enforcing a large-deviation–type bound for the incoming load, similar to a Chernoff bound. We call these adversaries *strongly bounded.*

Formally, let $\mathcal{A}$ be an adversary that is injecting load into the system. Let $I_t$ be the load injected by the adversary during time step $t$.

DEFINITION 2.1. *We say that $\mathcal{A}$ is a $(\lambda, w, p, M)$* stochastic adversary *(where we assume that $w$ is a positive integer) if the following conditions hold for any time $t$ and any event $\mathcal{H}$ determined entirely by information about the system at or before time $t$.*

1. $\mathbf{E}\left[\sum_{i=1}^{w} I_{t+i} \,|\, \mathcal{H}\right] \le \lambda n w$.
2. $\mathbf{E}\left[\left(\sum_{i=1}^{w} I_{t+i}\right)^p \,|\, \mathcal{H}\right] \le M n^p w^p$.

*In addition, we say that $\mathcal{A}$ is* strongly bounded *if it also satisfies the following condition.*

3. *There is a constant $\alpha > 0$ and a constant $\beta \ge 1$ such that for any $\epsilon > 0$*

$$\mathbf{Pr}\left(\sum_{i=1}^{w} I_{t+i} > (1+\epsilon)\lambda n w \,|\, \mathcal{H}\right) \le e^{-\alpha\lambda n w \epsilon^{\beta}}.$$

Throughout the paper, we always assume that $\lambda$ and $p$ do not depend on $n$, while $w$ and $M$ may be functions of $n$.

We give a high-level description of the protocol here, deferring the details to section 3. After the generation of new load, the nodes execute a particular distributed randomized algorithm for choosing a random matching. The matching is not necessarily perfect, nor is it necessarily chosen uniformly from all possible matchings, although it does have some important properties that we will exploit later. Once the matching is chosen, every two matched nodes equalize their load (up to one token). For simplicity, we refer to this protocol as $\mathcal{P}$.

We first show that the system (using $\mathcal{P}$) is stable under the stochastic adversary model (for $\lambda < 1$ and $p > 2$). That is, the expected total load in the system is bounded with respect to time. The following theorem, proven in sections 5.1 and 5.3, relates the load in the system to the network topology and establishes the stability of the system. We assume a connected network $G = (V, E)$ that has $n$ nodes, maximum degree at most $\Delta$, and whose Laplacian[1] has smallest nontrivial eigenvalue $\Lambda$. For convenience, we define the quantity $\gamma = \Lambda/16\Delta$.

THEOREM 2.2. *Suppose that we run the system with a $(\lambda, w, p, M)$ adversary, where $\lambda < 1$ and $p > 2$, using protocol $\mathcal{P}$ (described in section 3). Let $L_t$ be the load of the system at time $t$. Then the system is stable and*

$$\limsup_{t\to\infty} \mathbf{E}[L_t \,|\, L_0] = O(\gamma^{-1} n (w + \ln n)(1+M)^{3p}) \qquad \textit{as } n \to \infty.$$

*In addition, if the adversary is strongly bounded, then for any constant $c > 0$ there is a constant $\kappa = \kappa(c)$, such that for sufficiently large $n$,*

$$\liminf_{t\to\infty} \mathbf{Pr}(L_t \le \kappa\gamma^{-1} n (w + \ln n)) \ge 1 - n^{-c}.$$

*The above bounds hold without the limits if the system starts with no load.*

[1]Let $A$ denote the adjacency matrix of a graph $G$, and let $D = (d_{ij})$, where $d_{ij}$ is the degree of node $i$ if $i = j$, and is 0 otherwise. The Laplacian of $G$ is the matrix $L = D - A$. The eigenvalues of $L$ are $0 = \Lambda_1 \le \Lambda_2 \le \cdots \le \Lambda_n$, and $\Lambda_2 = \Omega(n^{-2})$ if $G$ is connected.

Next we address the efficiency of the load balancing protocol, an important performance parameter that was not addressed in most previous work. That is, we relate the waiting time of jobs to the topology of the network. Since protocol $\mathcal{P}$ treats all tokens equally regardless of their ages, it cannot guarantee efficient delivery time for individual packets. To bound the waiting time of jobs in the system, we augment the protocol with a distributed version of the *first-in-first-out* queueing discipline, requiring that a node always respect the ages of its tokens in the load balancing step, and always consume its oldest token in the load consumption step (see section 3 for details). We denote this version of $\mathcal{P}$ by $\mathcal{P}^*$.

THEOREM 2.3. *Suppose that we run the system with a $(\lambda, w, p, M)$ adversary, where $\lambda < 1$ and $p > 2$, using protocol $\mathcal{P}^*$ (described in section 3). Let $W_t$ be the wait of a job that arrived at time $t$. Then*

$$\limsup_{t\to\infty} \mathbf{E}[W_t \mid L_0] = O(\gamma^{-1}(w + \ln n)(1+M)^{3p}) \qquad \text{as } n \to \infty.$$

*In addition, if the adversary is strongly bounded, then for any constant $c > 0$ there is a constant $\kappa = \kappa(c)$, such that for sufficiently large $n$,*

$$\liminf_{t\to\infty} \mathbf{Pr}(W_t \le \kappa\gamma^{-1}(w + \ln n)) \ge 1 - n^{-c}.$$

*The above bounds hold without the limits if the system starts with no load.*

In particular, for bounded degree regular expanders, $\gamma = \Theta(1)$, and the diameter of the graph is $\Theta(\ln n)$, so for these graphs the above result is optimal.

As a special case of a strongly bounded adversary, we consider the generator model that appeared in [4]. We contrast our results with the work stealing load balancing protocol that is analyzed there and show that in sparse networks, both the load in the system and the waiting time of a job can be exponential in the network size.

Notice that both Theorems 2.2 and 2.3 are concerned with the long-term behavior of the system. In both cases, we show that if certain restrictions are placed on the adversary, then the system behaves well most of the time. Since we focus on the asymptotic behavior of the system, every valid system state will occur infinitely often. Therefore the analysis must ensure that the system rarely enters undesirable states and quickly recovers from them when it does. Simple probabilistic techniques (e.g., estimating the probability of rare events for every time point and using a union bound over all time steps) are not sufficient to achieve these goals. We require more sophisticated arguments.

There is a substantial class of results, based on modeling the system's evolution as a Markov chain, that is frequently employed in these sorts of analyses—the proof of stability for the work stealing protocol in [4] is one example. Furthermore, for this Markov chain setting, there are additional tools for proving rapid convergence to a stationary distribution (see Meyn and Tweedie [12]), as well as for deriving properties of the stationary distribution (see [4] and the references therein). However, many of these results are qualitative and not quantitative. More importantly, they restrict the class of problems that they can model. In the particular context of adversarial load balancing that is the focus of this work, any straightforward application of such tools will only yield results for Markovian adversaries (i.e., adversaries whose usage allows for the system to be modeled as a time-homogeneous Markov chain)—the analysis in [4] is an example. While it is not obvious whether this restricted class of adversaries is really any weaker than the general class presented earlier, it is clear that results in the general adversarial model serve as a compelling argument for the efficacy of the load balancing protocol under a wide range of input conditions.

Since we seek greater generality than Markov chain results are known to provide, we need more elaborate tools. In the conference version of this paper [2], we apply results from renewal theory to show that the system has efficient long-term behavior under a particular non-Markovian adversary. In the current work, we generalize the results presented there by analyzing the behavior under the more general adversaries of Definition 2.1. To this end, we show that the load in the system behaves like a supermartingale above some threshold and then employ a result of Pemantle and Rosenthal [17] for the analysis of such processes. Finally, in section 5.3, we derive stronger, high-probability results for the more restricted adversaries.

**3. Protocol.** If we were simply interested in a stability result we could use the protocol studied in [8]: in each step nodes are matched randomly with adjacent neighbors in the network, and if node $v$ is matched with node $u$, they equalize their load subject to integer rounding. The details of the protocol (which we call protocol $\mathcal{P}$) are given below.

```
1. Matching phase:
   • For each node $i$
      – Node $i$ inserts each incident edge $(i,j)$ into a set $S$ with probability $\frac{1}{8\max(d_i,d_j)}$, where
        $d_i$ is the degree of node $i$.
      – Node $i$ removes edge $(i,j)$ from $S$ if some edge $(i,k)$ or $(j,k)$ is in $S$, with $k \neq i,j$.
   • Let the matching $M$ consist of the remaining edges in $S$.
2. Transfer phase:
   • If $(i,j) \in M$
      – $i$ and $j$ equalize their loads so that, say, $i$ gets load $\lceil(\ell_t(i)+\ell_t(j))/2\rceil$ and $j$ gets load
        $\lfloor(\ell_t(i)+\ell_t(j))/2\rfloor$, where $\ell_t(i)$ is the load of processor $i$ in the beginning of the step.
```

To bound the waiting time of jobs in the system we need to augment the above protocol with a distributed version of the *first-in-first-out* queueing discipline. It is not enough to require that a node consume the oldest job in its queue; we also need to consider the jobs' ages in the load balancing procedure. In particular, when $u$ and $v$ are matched they should not only equalize their total load but also equalize (up to rounding) the load that they have above any given age. A simple method to maintain this property is given by protocol $\mathcal{P}^*$ below. Note that protocol $\mathcal{P}^*$ is a special case of protocol $\mathcal{P}$.

```
1. Matching phase:
   • For each node $i$
      – Node $i$ inserts each incident edge $(i,j)$ into a set $S$ with probability $\frac{1}{8\max(d_i,d_j)}$, where
        $d_i$ is the degree of node $i$.
      – Node $i$ removes edge $(i,j)$ from $S$ if some edge $(i,k)$ or $(j,k)$ is in $S$, with $k \neq i,j$.
   • Let the matching $M$ consist of the remaining edges in $S$.
2. Transfer phase:
   • If $(i,j) \in M$
      – Let $J^i_1, J^i_2, \ldots, J^i_{\ell_t(i)}$ (where $\ell_t(i)$ is the load of processor $i$ in the beginning of the step),
        and let $J^j_1, J^j_2, \ldots, J^j_{\ell_t(j)}$ be the jobs in the queues of nodes $i$ and $j$, respectively, sorted
        from oldest to newest.
      – Node $i$ sends jobs $J^i_2, J^i_4, \ldots, J^i_{2\lfloor \ell_t(i)/2\rfloor}$ to node $j$. Similarly, node $j$ sends jobs
        $J^j_2, J^j_4, \ldots, J^j_{2\lfloor \ell_t(i)/2\rfloor}$ to node $i$.
      – Node $i$ merges jobs $J^i_1, J^i_3, J^i_5, \ldots$ with $J^j_2, J^j_4, J^j_6, \ldots$ in its queue, so that, finally, if a
        job $J$ is older than a job $J'$, then job $J$ is in the queue before job $J'$.
      – Similarly, node $j$ merges jobs $J^j_1, J^j_3, J^j_5, \ldots$ with $J^i_2, J^i_4, J^i_6, \ldots$ in its queue, so that,
        finally, if a job $J$ is older than a job $J'$, then job $J$ is in the queue before job $J'$.
```

**4. Analysis of the static case.** We first analyze our load balancing protocol in a static setting in which some initial load is placed on the $n$ processors, and load is moved between processors until the loads in all the processors are approximately equal. No new load is added to or removed from the system throughout the execution of the protocol.

Our analysis of the static case is based on coupling the execution of our protocol with the nonintegral protocol studied in [8]. The only difference between the two protocols is that in the nonintegral protocol the load is equally distributed between the two matched processors with no rounding. We consider two copies of the network starting with the same initial distribution, one using our protocol and the other using the nonintegral protocol. The two processes are coupled so that they use the same matching at every time step.

Fix any initial distribution of $K$ tokens to the nodes in $V$, and let $\bar{\ell} = K/n$. For each time step $t \geq 0$ and $u \in V$, let $\ell_t(u)$ be the number of tokens at $u$ at the end of step $t$ of the original, integral protocol, and let $\ell'_t(u)$ be the number of tokens at $u$ at the end of step $t$ in the nonintegral copy of the protocol. Also, for each time step $t \geq 0$, let $\Phi'_t = \sum_{v \in V}(\ell'_t(v) - \bar{\ell})^2$. Note that if the total load in the system is $K$, then for any $t \geq 0$, $\Phi'_t \leq K^2$. Notice that $\Phi'_t$ corresponds to the variance of the load on the nodes, and it is easy to see that it is nonincreasing with time, which, intuitively, means that successive applications of the load balancing protocol even out the distribution of the tokens to the nodes.

Recall that $\gamma = \Lambda/16\Delta$. The performance of the nonintegral protocol was analyzed in terms of $\gamma$ in [8]. The relevant result is the following lemma, which follows immediately from Theorem 1 in that work.

LEMMA 4.1. *For any $t \geq 0$,*

$$\mathbf{E}[\Phi'_{t+1} \,|\, \Phi'_t] \leq (1-\gamma)\Phi'_t.$$

Adapting the technique in [8] we can prove the following static load balancing result for the nonintegral copy.

LEMMA 4.2. *If $t \geq \gamma^{-1}(2\ln K + c\ln n)$, then the probability that there is some $v \in V$ with $|\ell'_t(v) - \bar{\ell}| > 1$ is at most $1/n^c$.*

*Proof.* By Lemma 4.1 we get

$$\mathbf{E}[\Phi'_t] = \mathbf{E}[\,\mathbf{E}[\Phi'_t \,|\, \Phi'_{t-1}]\,] \leq (1-\gamma)\mathbf{E}[\Phi'_{t-1}].$$

By applying the same argument $t$ times we get

$$\begin{aligned}\mathbf{E}[\Phi'_t] &= (1-\gamma)^t \Phi'_0 \\ &\leq \Phi'_0 e^{-\gamma t} \\ &\leq e^{-c\ln n},\end{aligned}$$

where in the last inequality we used the facts that $\Phi'_0 \leq K^2$ and $t \geq \gamma^{-1}(2\ln K + c\ln n)$. Applying Markov's inequality yields $\mathbf{Pr}(\Phi'_t > 1) < n^{-c}$. □

We will now tie the performance of our protocol to the performance of the nonintegral copy.

LEMMA 4.3. *For any $t \geq 0$ and any $v \in V$, $|\ell_t(v) - \ell'_t(v)| \leq t/2$, regardless of the chosen matchings and the rounding decisions in the original protocol.*

*Proof.* The proof is by induction on $t \geq 0$. If $t = 0$, then $\ell_t(v) = \ell'_t(v)$ for every $v \in V$, because no randomness has been introduced into the process yet. For the induction step, suppose that the lemma holds for time $t$. Choose any $v \in V$. If $v$ is not matched at time $t+1$, then $\ell_{t+1}(v) = \ell_t(v)$ and $\ell'_{t+1}(v) = \ell'_t(v)$, so the lemma holds by the induction hypothesis. Otherwise, $v$ is matched to some vertex $u$ at time $t+1$. In this case, for any rounding choice, $(\ell_t(u)+\ell_t(v)-1)/2 \le \ell_{t+1}(v) \le (\ell_t(u)+\ell_t(v)+1)/2$. Also, $\ell'_{t+1}(v) = (\ell'_t(u) + \ell'_t(v))/2$. These observations give

$$\begin{aligned}
|\ell_{t+1}(v) - \ell'_{t+1}(v)| &\overset{(a)}{\le} \frac{1}{2} + \left| \frac{\ell_t(u)+\ell_t(v)}{2} - \frac{\ell'_t(u)+\ell'_t(v)}{2} \right| \\
&\overset{(b)}{\le} \frac{1}{2} + \frac{1}{2}|\ell_t(u) - \ell'_t(u)| + \frac{1}{2}|\ell_t(v) - \ell'_t(v)| \\
&\overset{(c)}{\le} \frac{1}{2}\cdot\frac{t}{2} + \frac{1}{2}\cdot\frac{t}{2} + \frac{1}{2} \\
&= \frac{t+1}{2},
\end{aligned}$$

where (a) follows from previous observations, (b) follows from the triangle inequality, and (c) follows from the induction hypothesis. □

Putting Lemmas 4.2 and 4.3 together yields the following theorem.

THEOREM 4.4. *Let $\hat{t} = \gamma^{-1}(2\ln K + c\ln n)$. Then, for any $t \ge \hat{t}$, the probability that there is some $v \in V$ with $|\ell_t(v) - \bar{\ell}| > 1 + \frac{\hat{t}}{2}$ is at most $n^{-c}$.*

*Proof.* We first prove the theorem for the case $t = \hat{t}$. Regardless of the matchings generated in the first $\hat{t}$ steps, $|\ell_{\hat{t}}(v) - \ell'_{\hat{t}}(v)| \le \hat{t}/2$ for every $v \in V$ by Lemma 4.3. With probability at least $1 - 1/n^c$, $|\ell'_{\hat{t}}(v) - \bar{\ell}| \le 1$ for all $v \in V$ by Lemma 4.2. Adding these inequalities proves the theorem for the case $t = \hat{t}$. To extend the result for the case $t > \hat{t}$, notice that the sequence $\{\max_{v\in V}|\ell_t(v) - \bar{\ell}|\}_{t\ge\hat{t}}$ is nonincreasing. □

**5. Analysis of the dynamic case.** In order to analyze the long-term performance of the system, we split the time into *epochs* of a fixed length $T_E$ (to be defined later). We analyze each epoch in Theorem 5.1, which is the key ingredient in showing the stability and waiting-time properties of the system. Notice that the first part of the theorem, which we will apply for the stability result, holds for any protocol obeying the rules of $\mathcal{P}$, while the second part, which will be applied for the waiting-time guarantees, uses protocol $\mathcal{P}^*$.

THEOREM 5.1. *For any constant $c > 0$ and load $\Theta = \Theta_n > 0$, consider an epoch of length $T_E = T_D + T_C$, such that*

$$T_D \ge \gamma^{-1}(2\ln\Theta + c\ln n) \qquad \textit{and} \qquad T_C \ge \frac{\Theta}{n} + \frac{T_D}{2} + 1.$$

1. *Running protocol $\mathcal{P}$, if at time $\tau$ the load is $L_\tau$, then the system consumes at least $\min\{L_\tau, \Theta\}$ tokens in the next $T_E$ steps with probability at least $1 - n^{-c}$.*
2. *Running protocol $\mathcal{P}^*$, if at time $\tau$ the load is bounded by $\Theta$, then with probability at least $1 - n^{-c}$, all the jobs that exist in the system at time $\tau$ will be consumed by time $\tau + T_E$.*

*Proof.* For the purpose of the analysis we split the epoch into two parts. In the first $T_D$ steps we focus on the distribution of load between the processors, and in the remaining $T_C$ steps we focus on the consumption of load by the processors (although load is consumed throughout the whole execution by processors that have load).

We start by proving the first part of the theorem; a modification of that argument gives the second part. To analyze the distribution of load between the processors, we couple the actual execution of the protocol in the first $T_D$ steps with an execution of the protocol in a static setting that starts with a total load of exactly $\Theta$ and does not generate or consume any jobs. We refer to the actual execution of the protocol as the *dynamic copy* and the static execution as the *static copy*.

To formulate the coupling we color all the tokens (jobs) in the dynamic copy at time $\tau$ by red and blue. A subset of $M = \min\{L_\tau, \Theta\}$ tokens in the system at time $\tau$ is colored red, and the rest are colored blue. We now place $\Theta$ tokens in the static copy so that each node in the static copy starts the process with at least as many tokens as the number of red tokens in the corresponding node of the dynamic copy. New tokens that arrive through the execution of the dynamic copy are colored blue.

The executions of the two copies are coupled so that they use the same matching in each step and the same rounding decisions. When we equalize (up to one) the load between two vertices in the dynamic execution, we also equalize (up to one, using the same rounding rule) the number of red tokens in the two nodes, keeping the total number of red tokens in the two nodes as before (this can be achieved by recoloring some tokens). Finally, after each matching we recolor the tokens in the nodes again, preserving the total number of red tokens in each queue, but putting all the red tokens in a queue ahead of all the blue tokens.

LEMMA 5.2. *At any time $\tau \le t \le \tau + T_D$ the number of red tokens in each node of the dynamic copy is bounded by the number of tokens in the corresponding node in the static copy.*

*Proof.* We begin with some intuition. Initially the coloring of the tokens in the dynamic copy is such that the claim holds. New tokens that enter the system in the dynamic copy are colored blue, while no tokens in the static copy are removed. Furthermore, the matching operations in both copies are coupled, so we expect that if one node in the static copy gives half of its tokens to a neighbor, the same node in the dynamic copy will do the same with its red tokens. Therefore, we expect that the claim will hold at the next step. By induction, the lemma follows.

We now proceed formally by induction on $t$. The claim is true for $t = \tau$ by the construction of the static copy. Assume that the claim holds after the execution of step $t - 1$, and consider the load of node $u$ after the execution of step $t$. If $u$ was not part of a matching in step $t$, then the load of the static copy did not change. The number of red tokens of the dynamic copy either did not change or was reduced by 1 if a red token was consumed. In both cases, using the induction hypothesis, the number of red tokens in the dynamic copy after the execution of step $t$ is bounded by the number of tokens in the static copy of $u$.

Assume now that node $u$ was matched with node $v$ during step $t$. Recall that every time step in the process can be decomposed into three substeps: the insertion substep (where new load is created in the dynamic copy), the balancing substep (where the matching is chosen and the nodes in both copies equalize their loads), and the consumption substep (where the nonempty nodes in the dynamic copy consume a job). With this in mind, we define some new variables:

- $\ell_t^i(u)$, $\ell_t^b(u)$, $\ell_t^c(u)$ are the total number of tokens at node $u$ in the dynamic copy immediately following the insertion, balancing, and consumption substeps of time step $t$, respectively.
- $r_t^i(u)$, $r_t^b(u)$, $r_t^c(u)$ are the number of red tokens at node $u$ in the dynamic copy at that time.
- $s_t^i(u)$, $s_t^b(u)$, $s_t^c(u)$ are the number of tokens at node $u$ in the static copy at that time.

We analyze each substep separately.

• Initially, by the induction hypothesis, we have

$$r^c_{t-1}(u) \le s^c_{t-1}(u), \qquad r^c_{t-1}(v) \le s^c_{t-1}(v). \tag{5.1}$$

• After the insertion substep, we color the new tokens blue so the number of red tokens remains the same:

$$r^i_t(u) = r^c_{t-1}(u), \qquad r^i_t(v) = r^c_{t-1}(v). \tag{5.2}$$

The static copy does not accept new tokens, so

$$s^i_t(u) = s^c_{t-1}(u), \qquad s^i_t(v) = s^c_{t-1}(v). \tag{5.3}$$

So from relations (5.1), (5.2), and (5.3) we get that

$$r^i_t(u) \le s^i_t(u), \qquad r^i_t(v) \le s^i_t(v). \tag{5.4}$$

Notice also that the number of red tokens is bounded by the total number of tokens, so

$$r^i_t(u) \le \ell^i_t(u), \qquad r^i_t(v) \le \ell^i_t(v). \tag{5.5}$$

• After the balancing substep, we assume, without loss of generality, that the rounding is such that

$$\ell^b_t(u) = \left\lceil \frac{1}{2}(\ell^i_t(u) + \ell^i_t(v)) \right\rceil, \qquad \ell^b_t(v) = \left\lfloor \frac{1}{2}(\ell^i_t(u) + \ell^i_t(v)) \right\rfloor. \tag{5.6}$$

The results of the analysis of the static case in section 4 hold for an arbitrary rounding procedure. Thus, in the static copy we perform the rounding so that

$$s^b_t(u) = \left\lceil \frac{1}{2}(s^i_t(u) + s^i_t(v)) \right\rceil, \qquad s^b_t(v) = \left\lfloor \frac{1}{2}(s^i_t(u) + s^i_t(v)) \right\rfloor. \tag{5.7}$$

Finally, we recolor the tokens in the dynamic copy (we swap colors between some tokens), so that

$$r^b_t(u) = \left\lceil \frac{1}{2}(r^i_t(u) + r^i_t(v)) \right\rceil, \qquad r^b_t(v) = \left\lfloor \frac{1}{2}(r^i_t(u) + r^i_t(v)) \right\rfloor. \tag{5.8}$$

Then, using relation (5.4), we get

$$r^b_t(u) = \left\lceil \frac{1}{2}(r^i_t(u) + r^i_t(v)) \right\rceil \le \left\lceil \frac{1}{2}(s^i_t(u) + s^i_t(v)) \right\rceil = s^b_t(u) \tag{5.9}$$

and

$$r^b_t(v) = \left\lfloor \frac{1}{2}(r^i_t(u) + r^i_t(v)) \right\rfloor \le \left\lfloor \frac{1}{2}(s^i_t(u) + s^i_t(v)) \right\rfloor = s^b_t(v). \tag{5.10}$$

Notice that

$$r^b_t(u) + r^b_t(v) = \left\lceil \frac{1}{2}(r^i_t(u) + r^i_t(v)) \right\rceil + \left\lfloor \frac{1}{2}(r^i_t(u) + r^i_t(v)) \right\rfloor = r^i_t(u) + r^i_t(v),$$

which means that we haven't changed the number of red tokens, and notice also that by using relation (5.5) we get that

$$r_t^b(u) = \left\lceil \frac{1}{2}(r_t^i(u) + r_t^i(v)) \right\rceil \leq \left\lceil \frac{1}{2}(\ell_t^i(u) + \ell_t^i(v)) \right\rceil = \ell_t^b(u)$$

and

$$r_t^b(v) = \left\lfloor \frac{1}{2}(r_t^i(u) + r_t^i(v)) \right\rfloor \leq \left\lfloor \frac{1}{2}(\ell_t^i(u) + \ell_t^i(v)) \right\rfloor = \ell_t^b(v),$$

which ensure that the recoloring is valid (i.e., for both $u$ and $v$, the number of the red tokens is not more than the total number of tokens).

Finally, we recolor the tokens again (preserving the number of red tokens at each node), so that every red token in a queue is ahead of every blue token in that queue.

• After the consumption substep, since some red tokens may be consumed, we get that

$$r_t^c(u) \leq r_t^b(u), \qquad r_t^c(v) \leq r_t^b(v). \tag{5.11}$$

In the static copy there are no tokens being consumed, so

$$s_t^c(u) = s_t^b(u), \qquad s_t^c(v) = s_t^b(v). \tag{5.12}$$

Hence, from relations (5.9), (5.10), (5.11), and (5.12) we can finally prove the induction hypothesis for step $t$:

$$r_t^c(u) \leq s_t^c(u), \qquad r_t^c(v) \leq s_t^c(v). \qquad \square$$

We now turn to studying the consumption phase. Applying Theorem 4.4 to the execution of the static copy, we see that at time $\tau + T_D$ (the reader can verify that the condition of Theorem 4.4 holds for $t = T_D$ and $K = \Theta$), with probability at least $1 - n^{-c}$ no node of the static copy has more than $T_D/2 + \Theta/n + 1 \leq T_C$ tokens. Thus, by Lemma 5.2, with the same probability, no node in the dynamic copy has more than $T_C$ red tokens.

We continue to run the coupling from time $\tau + T_D$ with this new coloring. When balancing between two nodes, we recolor the tokens in exactly the same way as in the distribution phase (see Lemma 5.2), so that, in particular, if a node has some red tokens in its queue, then these tokens are at the front of the node's queue, before the blue tokens. In this case, notice that after a balancing substep, the maximum (over all the nodes in the graph) number of tokens does not increase. In the consumption substep the maximum (again, over all the nodes in the graph) number of red tokens decreases by 1, since the red tokens are at the front of their queues. Since initially at most $T_C$ red tokens are at a node with probability at least $1 - n^{-c}$, we conclude that with probability $1 - n^{-c}$, all the red tokens (which means at least $M = \min\{L_\tau, \Theta\}$ tokens) are consumed in this epoch. This completes the proof of the first part of the theorem.

The proof of the second part of the theorem is almost identical to the first one. We color initially all the $L_\tau \leq \Theta$ tokens of the dynamic copy at time $\tau$ red, while all the subsequent ones are colored blue, and again we consider the coupled static copy. Since now we are interested in the identities of the jobs that are being consumed, we do not allow recolorings of the tokens.

Notice though that protocol $\mathcal{P}^*$ ensures that during the whole epoch, if a node has both red and blue tokens in its queue, the red tokens are before the blue ones. Moreover, the transfer phase of $\mathcal{P}^*$ ensures that when node $u$ is matched with node $v$, they balance their red tokens (up to a difference of 1). Without loss of generality, we assume that the rounding is such that if the ensemble of the red tokens in the two nodes is odd, then node $u$ ends up with one more red token. Hence equations (5.8) remain true. We then perform the rounding in the static copy to ensure that equations (5.7) remain true as well.

As an aside, notice that it may be the case that after the balancing substep, node $v$'s *total* load is exactly one larger than $u$'s total load, in which case equations (5.6) may become

$$\ell_t^b(u) = \left\lfloor \frac{1}{2}(\ell_t^i(u) + \ell_t^i(v)) \right\rfloor, \qquad \ell_t^b(v) = \left\lceil \frac{1}{2}(\ell_t^i(u) + \ell_t^i(v)) \right\rceil.$$

Here, however, we analyze only the distribution of the red tokens, so the analysis is not affected by this fact.

Everything else is identical to the first part, and by the same reasoning we conclude that by the end of the distribution phase, for every node $u$, the number of red tokens in the dynamic copy is bounded by the number of tokens in the static copy, with probability at least $1 - n^{-c}$, which, by applying Theorem 4.4, implies that every node has at most $T_C$ red tokens.

Then, as in the first part, we can conclude that by the end of the consumption phase all the red tokens—which are exactly the tokens that were in the system in the beginning of the epoch—are consumed. ∎

An immediate consequence of the analysis of the second part of Theorem 5.1 is the following lemma, which gives a bound on the expected time needed until the initial load is distributed. We make use of the lemma in order to bound the expected waiting time.

LEMMA 5.3. *Assume that we are given $c, \Theta, T_D, T_C$, satisfying the conditions of Theorem 5.1, and assume that we balance according to protocol $\mathcal{P}^*$. If at time $\tau$ the load is bounded by $\Theta$, then the expected time needed until every node in the system has at most $T_C$ of the initial jobs is bounded by $2T_D$ for sufficiently large $n$. At every time, when a node has some of the initial load, this is at the head of its queue.*

*Proof.* The analysis is an extension of the proof of the second part of Theorem 5.1. We color the initial load at time $\tau$ red, all the new incoming load blue, and we let $T$ be the time until every node in the network has at most $T_C$ red tokens. We perform the same coupling as in Theorem 5.1 from time $\tau$ until time $\tau + T$. During this whole period the number of red tokens in the dynamic copy is bounded by the number of tokens in the static copy. Since the red tokens are the oldest tokens in the system, protocol $\mathcal{P}^*$ ensures that if a node has some red tokens and some blue tokens, the red tokens are always in front of the blue ones.

Thus, by the proof of Theorem 5.1, part 2, we get that $T \le T_C$ with probability at least $1 - n^{-c}$. It follows that $\lceil T/T_D \rceil$ is stochastically dominated by a geometric random variable with parameter $1 - n^{-c}$, so

$$\mathbf{E}[T] \le \frac{1}{1 - n^{-c}} T_D \le 2T_D$$

for sufficiently large $n$. ∎

**5.1. Stability.** In this section we prove the stability of the system under a $(\lambda < 1, w, p > 2, M)$ stochastic adversary. The main technical tool that we use is the following theorem, which follows immediately from [17, Corollary 2].

THEOREM 5.4. *Let $X_1, X_2, \ldots$ be a sequence of nonnegative random variables satisfying the following conditions:*

1. *There exist positive numbers $\alpha = \alpha_n$ and $\Theta = \Theta_n$ such that for all $x_1, \ldots, x_i$ with $x_i > \Theta$,*

$$\mathbf{E}[X_{i+1} - X_i \,|\, X_1 = x_1, \ldots, X_i = x_i] \le -\alpha.$$

2. *There exists a positive number $\xi = \xi_n$ and a $p = p_n > 2$ such that for all $x_1, \ldots, x_i$*

$$\mathbf{E}[|X_{i+1} - X_i|^p \,|\, X_1 = x_1, \ldots, X_i = x_i] \le \xi.$$

*Then there exists $\Xi = \Xi(X_0, \alpha, \Theta, \xi)$ and $t_0$ such that for all $t \ge t_0$,*

$$\mathbf{E}[X_t \,|\, X_0] \le \Xi + \max(0, X_0 - \Theta).$$

*Furthermore, assuming that $p$ is a constant with respect to $n$,*

$$\Xi = O\left(\Theta + \alpha\left(1 + \frac{\xi}{\alpha^p}\right)^{3p}\right) \qquad \textit{as } n \to \infty.$$

Our stability result is summarized in the following theorem.

THEOREM 5.5. *Suppose that we run protocol $\mathcal{P}$ with a $(\lambda, w, p, M)$ adversary, where $\lambda < 1$ and $p > 2$. Then*

$$\sup_{t \ge 0} \mathbf{E}[L_t \,|\, L_0] = O(\max(\gamma^{-1} n(w + \ln n)(1 + M)^{3p}, L_0)) \qquad \textit{as } n \to \infty.$$

*Proof.* We first present the high-level idea of the proof. Recall that $L_t$ is the load of the system at time $t$. We partition the time into *epochs* of some length $d$ and apply Theorem 5.4 to the subsequence $\{L_{di} \mid i \ge 0\}$. Using Theorem 5.1, we show that if the load at the beginning of an epoch is above some threshold $\Theta$, then the expected load at the end of that epoch is strictly smaller by a significant amount, proving condition 1 of Theorem 5.4. We then derive the required bound on the $p$th moment $L_{(i+1)d} - L_{id}$, establishing condition 2 of Theorem 5.4. Applying Theorem 5.4 then gives the desired bound on the maximum expected load at the end of epochs. Finally, we generalize to steps that are not multiples of $d$.

Let us now formalize the above argument. The two conditions that we want to satisfy are

$$\mathbf{E}[L_{(i+1)d} - L_{id} \,|\, L_0 = l_0, L_d = l_d, \ldots, L_{(i-1)d} = l_{(i-1)d}, L_{id} = \ell_{id} > \Theta] \le -\alpha,$$
$$\mathbf{E}[|L_{(i+1)d} - L_{id}|^p \,|\, L_0 = l_0, L_d = l_d, \ldots, L_{(i-1)d} = l_{(i-1)d}, L_{id} = \ell_{id}] \le \xi$$

for all $(l_0, l_1, \ldots, l_{id})$. In order to simplify the notation we denote

$$\mathcal{L}_{i-1} = \{L_0 = l_0, L_d = l_d, \ldots, L_{(i-1)d} = l_{(i-1)d}\},$$

so the conditions become

$$\mathbf{E}[L_{(i+1)d} - L_{id} \,|\, \mathcal{L}_{i-1}, L_{id} = \ell_{id} > \Theta] \le -\alpha, \tag{5.13}$$

$$\mathbf{E}[|L_{(i+1)d} - L_{id}|^p \,|\, \mathcal{L}_{i-1}, L_{id}] \le \xi. \tag{5.14}$$

Let

$$\Theta = \sigma\gamma^{-1}n(w + \ln n) \tag{5.15}$$

for some constant $\sigma$ that we will fix later. Also let

$$T_D = \gamma^{-1}(2\ln\Theta + \ln n) \qquad \text{and} \qquad T_C = \frac{\Theta}{n} + \frac{T_D}{2} + 1,$$

so that

$$\begin{aligned} T_E &= T_D + T_C \\ &= \frac{\Theta}{n} + 3\gamma^{-1}\ln\Theta + \frac{3}{2}\gamma^{-1}\ln n + 1 \\ &= \sigma\gamma^{-1}(w + \ln n) + 3\gamma^{-1}\ln\sigma + 3\gamma^{-1}\ln\gamma^{-1} + 3\gamma^{-1}\ln(w + \ln n) + \frac{9}{2}\gamma^{-1}\ln n + 1. \end{aligned} \tag{5.16}$$

We let $\sigma$ be such that $T_E/w$ is an integer, say $k$ (we show later that such a $\sigma$ exists), so that $T_E$ is a multiple of the window size $w$, and define the epoch length

$$d = kw = T_E. \tag{5.17}$$

Notice that $d = O(\gamma^{-1}(w + \ln n))$, a fact that we use later. Fix some time $t = id$ and consider what happens in the case that $L_t > \Theta$. By Theorem 5.1, part 1 (for $c = 1$), we get that within $d = T_E$ steps the system consumes at least $\Theta$ units of load, with probability at least $1 - 1/n$. Thus, the expected number of tokens consumed between steps $id$ and $(i+1)d$ is at least

$$\left(1 - \frac{1}{n}\right) \cdot \Theta = \sigma\gamma^{-1}n(w + \ln n) - \sigma\gamma^{-1}(w + \ln n), \tag{5.18}$$

independently of the past, and of any of the adversary's decisions.

Since an epoch consists of $k$ windows, by the definition of the adversary, the expected injected new load in the system from time $id$ until $(i+1)d$, conditioned on $\mathcal{L}_{i-1}$, is bounded by

$$k\lambda nw = \lambda nd.$$

Using (5.16) and (5.17) we can conclude that the expected new load injected by the adversary, conditioned on the history, is bounded by

$$\begin{aligned} \lambda nd &= \lambda\sigma\gamma^{-1}n(w + \ln n) + 3\lambda\gamma^{-1}n\ln\sigma \\ &\quad + 3\lambda\gamma^{-1}n\ln\gamma^{-1} + 3\lambda\gamma^{-1}n\ln(w + \ln n) + \frac{9}{2}\lambda\gamma^{-1}n\ln n + \lambda n \\ &\le \lambda\sigma\gamma^{-1}n(w + \ln n) + 3\lambda\gamma^{-1}n\ln\sigma \\ &\quad + 9\lambda\gamma^{-1}n\ln(c'n) + 3\lambda\gamma^{-1}n\ln(w + \ln n) + \frac{9}{2}\lambda\gamma^{-1}n\ln n + \lambda n \qquad (5.19) \\ &= \gamma^{-1}n(w + \ln n)\left[\lambda\sigma + \frac{3\lambda\ln\sigma}{w + \ln n} + \frac{9\lambda\ln(c'n)}{w + \ln n} + \frac{3\lambda\ln(w + \ln n)}{w + \ln n}\right. \\ &\qquad \left. + \frac{9\lambda\ln n}{2(w + \ln n)} + \frac{\lambda}{\gamma^{-1}(w + \ln n)}\right], \end{aligned}$$

where the inequality follows from the fact that $\gamma^{-1} \leq c'n^3$ for some constant $c'$, since $\Lambda = \Omega(n^{-2})$ for any connected graph. Therefore, from relations (5.18) and (5.19) we get

$$\begin{aligned}\mathbf{E}[L_{(i+1)d} - L_{id} \,|\, \mathcal{L}_{i-1}] &\leq -\sigma\gamma^{-1}n(w+\ln n) + \gamma^{-1}n(w+\ln n)\\ &\qquad \left[\lambda\sigma + \frac{3\lambda\ln\sigma}{w+\ln n} + \frac{9\lambda\ln(c'n)}{w+\ln n} + \frac{3\lambda\ln(w+\ln n)}{w+\ln n}\right.\\ &\qquad \left. + \frac{9\lambda\ln n}{2(w+\ln n)} + \frac{\lambda}{\gamma^{-1}(w+\ln n)} + \frac{\sigma}{n}\right]\\ &\leq -\sigma\gamma^{-1}n(w+\ln n) + \gamma^{-1}n(w+\ln n)(\lambda\sigma + Q),\end{aligned}$$

for sufficiently large $n$, where $Q$ is a constant independent of $\sigma$. Hence,

$$\mathbf{E}[L_{(i+1)d} - L_{id} \,|\, \mathcal{L}_{i-1}] \leq -\gamma^{-1}n(w+\ln n)(\sigma - \lambda\sigma - Q).$$

If

$$\sigma > \frac{Q}{1-\lambda},$$

then relation (5.13) is satisfied for sufficiently large $n$. We therefore let $\sigma$ be the smallest number that is greater than $Q/(1-\lambda)$ that ensures that $d = T_E$ is a multiple of the window size $w$. Notice that the fact that $d$ is a continuous function of $\sigma$ ensures that such a value of $\sigma$ exists and satisfies

$$\sigma \leq \frac{2Q}{1-\lambda}.$$

Thus condition 1 is satisfied for $\alpha = O(\gamma^{-1}n(w+\ln n))$.

We now turn our attention to relation (5.14). Let $J_i$ and $Z_i$ be the number of tokens injected by the adversary and consumed by the processors, respectively, during the $i$th epoch. We note that

$$\begin{aligned}|L_{(i+1)d} - L_{id}|^p &= |J_i - Z_i|^p\\ &\leq J_i^p + Z_i^p\\ &\leq J_i^p + d^p n^p,\end{aligned} \tag{5.20}$$

where the last inequality follows from the fact that the system (deterministically) consumes at most $n$ tokens in every step.

We now bound $\mathbf{E}[J_i^p \,|\, \mathcal{L}_{i-1}, L_{id}]$. Write $J_i = \sum_{j=0}^{k-1} Y_j$, where

$$Y_j = \sum_{t=0}^{w-1} I_{id+jw+t}$$

is the number of tokens injected by the adversary during the window $[id+jw,\, id+(j+1)w-1]$.

The second condition on the stochastic adversary gives, for all $j$,

$$\mathbf{E}[Y_j^p] \leq Mn^p w^p.$$

Applying Hölder's inequality gives

$$J_i^p = \left(\sum_{j=0}^{k-1} Y_j\right)^p \leq \left(\sum_{j=0}^{k-1} 1\right)^{p\left(1-\frac{1}{p}\right)} \left(\sum_{j=0}^{k-1} Y_j^p\right) = k^{p-1}\sum_{j=0}^{k-1} Y_j^p,$$

and therefore we get

$$\mathbf{E}[J_i^p \mid \mathcal{L}_{i-1}, L_{id}] \leq k^p M n^p w^p. \tag{5.21}$$

Therefore, relations (5.20) and (5.21) imply that relation (5.14) is satisfied with

$$\xi = k^p M n^p w^p + d^p n^p = (M+1) d^p n^p,$$

and by Theorem 5.4 we deduce that

$$\sup_{i \geq 0} \mathbf{E}[L_{id} \mid L_0] = O(\max(\gamma^{-1} n (w + \ln n)(1+M)^{3p}, L_0)).$$

We have now proven the theorem for $t$ corresponding to the beginning of an epoch. To finish the proof, we have for any $t \geq 0$

$$\begin{aligned}
\mathbf{E}[L_t \mid L_0] &= \mathbf{E}[L_{\lfloor t/d \rfloor d + (t - \lfloor t/d \rfloor d)} \mid L_0] \\
&\leq \mathbf{E}[L_{\lfloor t/d \rfloor d} \mid L_0] + \mathbf{E}\left[ \sum_{j = \lfloor t/d \rfloor d}^{t} I_j \,\middle|\, L_0 \right] \\
&\leq \sup_{i \geq 0} \mathbf{E}[L_{id} \mid L_0] + \lambda n d \\
&= O(\max(\gamma^{-1} n (w + \ln n)(1+M)^{3p}, L_0)). \qquad \square
\end{aligned}$$

THEOREM 5.6. *Given any initial load $L_0$, with probability 1 there is an $i \geq 0$, such that $L_{id} \leq \Theta$ for $\Theta$ defined as in (5.15).*

*Proof.* For intuition, assume that the initial load $L_0$ is above $\Theta$. Then by relation (5.13), we expect it to decrease below $\Theta$ after a sufficiently long time period. In order to prove this fact, we use martingale techniques. Since the expected load decreases independently of the past, we can couple the load with a supermartingale until it drops below $\Theta$. Then we can apply a martingale convergence theorem to show that the supermartingale (and therefore the coupled system load) will eventually reach $\Theta$.

Proceeding formally, we define a supermartingale $\{Y_{id} \mid i \geq 0\}$ with respect to the sequence $\{L_{id} \mid i \geq 0\}$, where

$$Y_{id} = \begin{cases} \max(L_0, \Theta) & \text{for } i = 0, \\ L_{id} & \text{if } L_{(i-1)d} > \Theta, \\ \Theta & \text{if } L_{(i-1)d} \leq \Theta. \end{cases}$$

As long as $Y_{(i-1)d} > \Theta$ the two sequences of random variables are identical. The sequence $L_{id}$ assumes a value $\leq \Theta$ if and only if there is an index $j$ such that $Y_{jd} = \Theta$.

The (nonnegative) supermartingale $Y_{id}$ converges with probability 1 to a random variable $Y$ (see [9, Theorem 5.1]), and since

$$\mathbf{E}[Y_{(i+1)d} \mid Y_{id} > \Theta] \leq Y_{id} - \alpha$$

for some $\alpha > 0$ (defined in the proof of Theorem 5.5), we have $\lim_{i \to \infty} \mathbf{E}[Y_{id}] = \Theta$. By applying Fatou's lemma [6, p. 110] we get $\mathbf{E}[Y] \leq \liminf_{i \to \infty} \mathbf{E}[Y_{id}] = \Theta$. Thus, with probability 1 the sequence $\{L_{id}\}$ assumes at some time $jd$ a value less than or equal to $\Theta$. $\square$

Corollary 5.7.

1. *If the system starts with no load, then for sufficiently large $n$,*

$$\sup_{t\geq 0} \mathbf{E}[L_t] = O(\gamma^{-1} n(w + \ln n)(1 + M)^{3p}).$$

2. *For any starting conditions*

$$\limsup_{t\to\infty} \mathbf{E}[L_t \mid L_0] = O(\gamma^{-1} n(w + \ln n)(1 + M)^{3p}) \qquad \textit{as } n \to \infty.$$

*Proof.* The first part follows from Theorem 5.5, while the second part uses also Theorem 5.6. □

**5.2. Waiting time.** Having established that the system is stable, the next important performance parameter is the waiting time of a job from the time it enters the system until it is executed. For a given task that enters the system at time $t$, let $W_t$ be the number of steps until the task is executed. Following the discussion of section 3, throughout this section we assume that we perform protocol $\mathcal{P}^*$.

Theorem 5.8. *Suppose that we run protocol $\mathcal{P}^*$ with a $(\lambda, w, p, M)$ adversary, where $\lambda < 1$ and $p > 2$. Then*

$$\sup_{t\geq 0} \mathbf{E}[W_t \mid L_0] = O(\max(\gamma^{-1}(w + \ln n)(1 + M)^{3p}, L_0/n)) \qquad \textit{as } n \to \infty.$$

*Proof.* We begin with some intuition. By the results of section 5.1, we expect the load $L_t$ at time $t$ to be low, namely, bounded by $O(\max(\gamma^{-1}n(w+\ln n)(1+M)^{3p}, L_0))$. We also expect that the distribution protocol will rapidly distribute this load among the nodes (even if it is not already distributed, and regardless of any load that comes in after time $t$)—this is formalized by Lemma 5.3. Once this load is evenly distributed, it can be quickly consumed, since $\mathcal{P}^*$ ensures that every node consumes the oldest token in its queue at the end of every time step. Therefore, the expected time to consume all the load is $O(\mathbf{E}[L_t]/n)$.

We now proceed formally. Assume that at some time $t$ the load in the system is $L_t$. We apply Lemma 5.3 with $c = 1$, $\Theta = L_t$, $T_D = \gamma^{-1}(2 \ln L_t + \ln n)$ and

$$T_C = \frac{\Theta}{n} + \frac{T_D}{2} + 1,$$

and we get that the expected time, conditioned on any past event $\mathcal{H}$, until all the tokens that were present at time $t$ are consumed (measured from time $t$) is at most

$$2T_D + T_C \leq 4T_C + T_C = 5T_C$$

for $n \geq 2$. Thus for $n \geq 2$,

$$\mathbf{E}[W_t \mid L_t, \mathcal{H}] \leq 5T_C = \frac{5L_t}{n} + \frac{5\gamma^{-1}}{2}(2 \ln L_t + \ln n) + 5.$$

Consequently, for sufficiently large $n$, we have that for any $t \geq 0$,

$$\begin{aligned}
\mathbf{E}[W_t \mid L_0] &= \sum_{\ell_t} \mathbf{E}[W_t \mid L_0, L_t = \ell_t] \cdot \mathbf{Pr}(L_t = \ell_t \mid L_0) \\
&\leq \sum_{\ell_t} \left( \frac{5\ell_t}{n} + \frac{5\gamma^{-1}}{2}(2 \ln \ell_t + \ln n) + 5 \right) \cdot \mathbf{Pr}(L_t = \ell_t \mid L_0) \\
&= \mathbf{E}\left[ \frac{5L_t}{n} + \frac{5\gamma^{-1}}{2}(2 \ln L_t + \ln n) + 5 \,\middle|\, L_0 \right] \\
&= \frac{5\,\mathbf{E}[L_t \mid L_0]}{n} + \frac{5\gamma^{-1}}{2}(2\,\mathbf{E}[\ln L_t \mid L_0] + \ln n) + 5 \\
&\leq \frac{5\,\mathbf{E}[L_t \mid L_0]}{n} + \frac{5\gamma^{-1}}{2}(2\, \ln \mathbf{E}[L_t \mid L_0] + \ln n) + 5 \\
&= O(\max(\gamma^{-1}(w + \ln n)(1 + M)^{3p}, L_0/n)),
\end{aligned}$$

where the second-to-last step follows from Jensen's inequality applied to the concave function $f(x) = \ln x$, and the last step follows from Theorem 5.5. $\square$

Applying Theorem 5.6 we have the following.

Corollary 5.9.

1. *If the system starts with no load, then for sufficiently large $n$,*

$$\sup_{t \geq 0} \mathbf{E}[W_t] = O(\gamma^{-1}(w + \ln n)(1 + M)^{3p}).$$

2. *For any starting conditions,*

$$\limsup_{t \geq 0} \mathbf{E}[W_t \mid L_0] = O(\gamma^{-1}(w + \ln n)(1 + M)^{3p}) \qquad \textit{as } n \to \infty.$$

*Proof.* The first part follows from Theorem 5.8, while the second part uses also Theorem 5.6. $\square$

**5.3. Strongly bounded adversaries.** Recall that a strongly bounded adversary satisfies the additional requirement that for some constants $\alpha > 0$, $\beta \geq 1$, for any $\epsilon > 0$, the probability that the total number of new jobs that arrive in a given interval of length $w$ is greater than $(1 + \epsilon)\lambda n w$ is bounded by $e^{-\alpha \lambda n w \epsilon^{\beta}}$.

In this subsection we strengthen the preceding results for adversaries that are strongly bounded. The Chernoff-type restrictions on the input stream allow us to get high-probability results for the load and the waiting time.

**5.3.1. High-probability bound on load.** Our first result bounds the probability that at a given time point the load of the system is high.

Theorem 5.10. *Consider a system where load is injected by a strongly bounded adversary. Let $L_t$ be the load of the system at time $t$. Then for any sufficiently large constant $c_1$ there exists a constant $c' > 0$ such that*

$$\limsup_{t \to \infty} \mathbf{Pr}(L_t > c_1 \gamma^{-1} n(w + \ln n)) \leq 3n^{-c'}.$$

*The above holds without the limit if the system starts with no load.*

*Proof.* Let $\Theta = c_1 \gamma^{-1} n(w + \ln n)$. We observe the system at some time $t$, and we need to bound the probability that the load at time $t$ is above $\Theta$. Therefore, we assume that the load at time $t$ is above $\Theta$ and calculate the probability of the events that may have led to such a load. If the load at some time $t' < t$ were smaller than $\Theta$ (which holds with probability 1 as $t \to \infty$ by Theorem 5.6), then from time $t'$ up to $t$ some rare events have taken place and increased the load much more than expected. We bound the probability of those events, thus bounding the probability that the load at time $t$ is above $\Theta$.

Similarly to Theorem 5.5, we split time into epochs of length

$$T_E = c_2\gamma^{-1}(w + \ln n)$$

starting from time $t$ and going backwards. The constant $c_2$ (which depends on $c_1$) is chosen so that the epoch length is a multiple $k$ of the window size ($T_E = kw$). Let $\mathcal{B}$ be the event $\{L_t \geq \Theta\}$, and for $i \leq t/T_E$ let $\mathcal{B}_i$ be the event that the load of the system is above $\Theta$ for exactly the last $i$ epochs. More precisely,

$$\mathcal{B}_i = \{\forall j = 0, \ldots, i-1 : \ L_{t-jT_E} \geq \Theta, \quad L_{t-iT_E} < \Theta\}.$$

Let $\mathcal{C}_t$ be the event that the load in the system was not always above $\Theta$ before time $t$. Formally,

$$\mathcal{C}_t = \{\exists t' \leq t : \ L_{t'} \leq \Theta\}.$$

Then we have

$$\mathbf{Pr}(\mathcal{B} \,|\, \mathcal{C}_t) = \sum_{i=1}^{\lfloor t/T_E \rfloor} \mathbf{Pr}(\mathcal{B}_i \mid \mathcal{C}_t). \tag{5.22}$$

To estimate $\mathbf{Pr}(\mathcal{B}_i \,|\, \mathcal{C}_t)$ we distinguish between two cases, depending on the total load injected by the adversary during the $i$ epochs immediately before $t$. Either the adversary inserted a lot of new jobs during this time, or he inserted a reasonable number of new jobs and the protocol failed to reduce the total load. Both cases are intuitively unlikely: the first by the strong bound on the adversary, and the second by the efficacy of the protocol. We bound the two cases separately and then use a union bound. We fix a constant $\epsilon > 0$ (whose actual value will be determined later) and define $\mathcal{M}$ as the event that "the injected load during the $i$ epochs immediately preceding $t$ is less than $K = (1+\epsilon)i\lambda n T_E = (1+\epsilon)i\lambda nkw$." Then we have

$$\mathbf{Pr}(\mathcal{B}_i \,|\, \mathcal{C}_t) \leq \mathbf{Pr}(\overline{\mathcal{M}} \,|\, \mathcal{C}_t) + \mathbf{Pr}(\mathcal{B}_i \cap \mathcal{M} \,|\, \mathcal{C}_t). \tag{5.23}$$

We bound each term separately, starting with $\mathbf{Pr}(\overline{\mathcal{M}} \,|\, \mathcal{C}_t)$. The first $K$ jobs can be distributed in the $ik$ windows of the period in

$$\binom{K + ik - 1}{ik} \tag{5.24}$$

ways.

We now bound the probability of each such distribution of the inserted jobs $(K_1, K_2, \ldots, K_{ik})$ (so that $\sum_{j=1}^{ik} K_j = K$). Recall that the expected number of jobs during the $j$th time window is at most $\lambda nw$. Define $\epsilon_j$ as the deviation of $K_j$ above $\lambda nw$, namely,

$$\epsilon_j = \max\left(0, \frac{K}{\lambda nw} - 1\right).$$

In other words, $\epsilon_j$ is such that

$$K_j = (1+\epsilon_j)\lambda n w$$

if $K > \lambda n w$ and $\epsilon_j = 0$ otherwise. Since $\sum K_j = K$, we get that

$$\sum_{j=1}^{ik} \epsilon_j \geq ik\epsilon.$$

By using the definition of the strongly bounded adversary, we can bound the probability (conditioned on any past event) that in the $j$th window at least $K_j$ were generated by

$$e^{-\alpha\lambda n w \epsilon_j^\beta}.$$

Therefore, the probability of a particular distribution $(K_1, \dots, K_{id})$ of the first $K$ jobs is bounded by

$$\prod e^{-\alpha\lambda n w \epsilon_j^\beta} = e^{-\alpha\lambda n w \sum \epsilon_j^\beta}.$$

Since $\beta \geq 1$, and $\sum \epsilon_j \geq ik\epsilon$, we obtain $\sum \epsilon_j^\beta \geq ik\epsilon^\beta$ (by raising to the $\beta$th power and using Hölder's inequality as in relation (5.21)), and the aforementioned probability becomes

$$e^{-\alpha\lambda n w ik \epsilon^\beta}.$$

Together with (5.24) we get that

$$\begin{aligned}
\mathbf{Pr}(\overline{\mathcal{M}} \,|\, \mathcal{C}_t) &\leq \binom{K+ik-1}{ik} e^{-\alpha\lambda n w ik\epsilon^\beta} \\
&< \binom{K+ik}{ik} e^{-\alpha\lambda n w ik\epsilon^\beta} \\
&\leq \left(\frac{(K+ik)e}{ik}\right)^{ik} e^{-\alpha\lambda n w ik\epsilon^\beta} \\
&= e^{ik\ln(K+ik)+ik-\alpha\lambda n w ik\epsilon^\beta - ik\ln(ik)} \\
&\leq e^{ik\ln[ik((1+\epsilon)\lambda n w+1)]+ik-\alpha\lambda n w ik\epsilon^\beta - ik\ln(ik)} \\
&= e^{ik\ln[(1+\epsilon)\lambda n w+1]+ik-\alpha\lambda n w ik\epsilon^\beta} \\
&< n^{-\kappa i}
\end{aligned} \tag{5.25}$$

for any constant $\kappa$ and sufficiently large $n$.

Next we bound $\mathbf{Pr}(\mathcal{B}_i \cap \mathcal{M} \,|\, \mathcal{C}_t)$. By Theorem 5.1, part 1, if at the beginning of an epoch the load of the system is at least $\Theta$, then the load decreases by at least $\Theta$ with probability at least $1 - n^{-c}$. This is the case for the last $i-1$ epochs. However, at time $t - iT_E$ the load of the system is below $\Theta$, while at time $t$ the load is above $\Theta$. Moreover we have assumed that the total new load is at most $K = (1+\epsilon)i\lambda n T_E$. These facts imply that the consumed load is less than $K$, which in turn implies that in fewer than

$$\frac{K}{\Theta} = \frac{(1+\epsilon)\lambda c_2}{c_1} i \triangleq \mu i$$

epochs the consumed load was more than $\Theta$. By making $c_1$ sufficiently large and $\epsilon$ sufficiently small, we can guarantee that $\mu < 1$. In this case, the probability that, among the $i-1$ epochs, fewer than $\mu i$ consumed at least $\Theta$ load can be bounded by

$$\begin{aligned}\mathbf{Pr}(\mathcal{B}_i \cap \mathcal{M} \,|\, \mathcal{C}_t) &\leq \binom{i-1}{(1-\mu)i} \left(\frac{1}{n^c}\right)^{(1-\mu)i} \\ &\leq \left(\frac{e(i-1)}{(1-\mu)i}\right)^{(1-\mu)i} \left(\frac{1}{n^c}\right)^{(1-\mu)i} \\ &= \left(n^{-c} \cdot \frac{e}{1-\mu} \cdot \frac{i-1}{i}\right)^{(1-\mu)i} \\ &\leq n^{-(1-\mu)(c-1)i}.\end{aligned} \tag{5.26}$$

By combining (5.23), (5.25), and (5.26) we get that

$$\mathbf{Pr}(\mathcal{B}_i \,|\, \mathcal{C}_t) \leq 2n^{-(1-\mu)(c-1)i}.$$

Finally, we estimate the probability that the load is above $\Theta$ at time $t$ using (5.22). If we make $c$ and $c_1$ sufficiently large, we get that $(1-\mu)(c-1) > 0$, so the sum converges and we get

$$\mathbf{Pr}(\mathcal{B} \,|\, \mathcal{C}_t) = \sum_{i=1}^{\lfloor t/T_E \rfloor} 2n^{-(1-\mu)(c-1)i} \leq \sum_{i=1}^{\infty} 2n^{-(1-\mu)(c-1)i} \leq 3n^{-(1-\mu)(c-1)}.$$

From Theorem 5.6 we have $\lim_{t\to\infty} \mathbf{Pr}(\mathcal{C}_t) = 1$, which gives the result. □

**5.3.2. Waiting time.** By Theorem 5.1, part 2, we get that if the load of the system is bounded by $\Theta$ at some particular time, then with probability at least $1 - n^{-c} \geq 1 - n^{-c'}$ all the load that was in the system at that time is consumed during the next $T_E$ steps. The limiting probability that the load of the system is above $\Theta$ is bounded by $3n^{-c'}$. Summing the failure probabilities proves the following.

THEOREM 5.11. *Consider a system whose load is injected by a strongly bounded adversary. Let $W_t$ be the wait of a job that arrived at time $t$.*

$$\liminf_{t\to\infty} \mathbf{Pr}(W_t \leq c_2 \gamma^{-1}(w + \ln n)) \geq 1 - 4n^{-c'}.$$

*The above holds without the limit if the system starts with no load.*

**6. Work stealing and the generator models.** In this section we compare our results to the stochastic analysis of the *work stealing* load balancing protocol in [4]. In the work stealing protocol, nodes initiate load balancing steps only when their queues are empty. Concretely, after the new load generation, and before the load consumption, if a node $u$ is empty, it selects a random neighbor, say $v$. If the load of $v$ is $\ell_t(v)$, then $v$ transfers half of its load to $u$, so that eventually $u$ has load $\lfloor \ell_t(v) \rfloor$, while $v$ ends up with load $\lceil \ell_t(v) \rceil$. Finally the nonempty nodes execute one job from their queue. The advantage of the work stealing protocol is that the total load balancing work in the network at any given time is proportional to the number of processors with empty queues. In particular, as long as all the processors are working, the network does not perform any load balancing. However, in this section we show that the work stealing protocol is either unstable, or stable with expected load exponential in $n/d$, where $d$ is the minimum degree of the network. In contrast, protocol $\mathcal{P}$ is stable with expected load polynomial in $n$ for any connected network topology.

We prove the gap in performance between the two protocols in the job-generator model introduced in [4]. We distinguish between two versions of this input model. In the deterministic generator model, an adversary places $\lambda n$ job-generators in the processors in an arbitrary fashion at the beginning of every step (having full information of the history of the system), and each generator adds one new job to the processor on which it is placed. In the stochastic job-generator model an adversary places $n$ generators in the processors in an arbitrary fashion at the beginning of every time step (again having full information of the history), and each generator adds one new job to the processor on which it is placed with probability at most $\lambda < 1$, independently of the other generators.

Notice that both of these adversaries are special cases of a strongly bounded $(\lambda, w, p, M)$ adversary, with $w = 1$, $p = 3$ and corresponding $M$ being 0 for the deterministic and a constant for the stochastic. The deterministic model is clearly strongly bounded, while a standard Chernoff bound establishes the claim for the stochastic model. Then by Theorems 2.2 and 2.3, we get that, in both cases, the load and waiting time are bounded (both in expectation and with high probability) by $O(\gamma^{-1} n \ln n)$ and $O(\gamma^{-1} \ln n)$, respectively.

We contrast these results with the performance of the work stealing load balancing protocol on similar input. Consider first a deterministic job-generator adversary that adds, in each step, up to $\lambda n$ tokens to the $n$-node network. Let $v$ be a node in the network with degree $d = o(n)$ (the claim is trivial for $d = \Omega(n)$, since then $n/d = O(1)$). In each step, the adversary puts $\lambda n - d$ tokens in $v$ and one token in each of its neighbors. Since the queues of the $d$ neighbors of $v$ are never empty, $v$ never participates in a load balancing step, and so its load increases (unboundedly) by $\lambda n - d - 1$ in each step.

Consider now the stochastic job-generator adversarial input process. Assume again a node $v$ with minimal degree $d = o(n)$, and assume that in each step the adversary places exactly $m = \lfloor n/(d+1) \rfloor$ generators at $v$ and at least $m$ generators at each of its neighbors. Let $\ell_t(v)$ denote the load at $v$ at time $t$.

Define $p = (2 + \lambda m/(1-\lambda))e^{-\lambda m}$ and assume that $n$ is large, so that $p < 1$ (we use this fact later). Fix some time step $t \geq 1$. If node $v$'s load did not increase during this time step, then either $v$ did not produce many jobs (at most 1) or one of $v$'s neighbors (say $u_i$, with degree $d_{u_i} \geq d$) was empty and chose to try to steal work from $v$, so $v$ gave away load. For the latter to happen, $u_i$ must have had no load (and in particular must have not produced any load at time $t$) and must have randomly chosen $v$ out of its $d_{u_i}$ neighbors. Thus,

$$\begin{aligned}\mathbf{Pr}(\ell_t(v) - \ell_{t-1}(v) < 1) &\leq (1-\lambda)^m + \lambda m (1-\lambda)^{m-1} + \sum_{i=1}^{d} \frac{1}{d_{u_i}} (1-\lambda)^m \\ &\leq p,\end{aligned} \tag{6.1}$$

conditioned on all past events.

We now construct a random sequence $\{\ell'_t(v)\}_{t \geq 0}$, and we show that it is stochastically dominated by the sequence $\{\ell_t(v)\}$, as follows. We have $\ell'_0(v) = 0$, and for $t > 0$,

$$\ell'_t(v) = \begin{cases} \ell'_{t-1}(v)/2 & \text{with probability } p, \\ \ell'_{t-1}(v) + 1 & \text{with probability } 1-p, \end{cases}$$

with all the random choices being independent. Notice that by relation (6.1) the load of node $v$ decreases with probability at most $p$. Moreover, if the load of node $v$ decreases, the new load will be at least half of the old load, and hence each load $\ell_t(v)$ stochastically dominates the corresponding $\ell'_t(v)$. Therefore we have $\mathbf{E}[\ell_t(v)] \geq \mathbf{E}[\ell'_t(v)]$.

Using the recursion

$$\mathbf{E}[\ell'_t(v)] = \frac{p}{2}\mathbf{E}[\ell'_{t-1}(v)] + (1-p)(\mathbf{E}[\ell'_{t-1}(v)] + 1),$$

we compute

$$\mathbf{E}[\ell'_t(v)] = \frac{2(1-p)}{p}\left[1 - \left(1 - \frac{p}{2}\right)^t\right]$$

and

$$\begin{aligned}\mathbf{E}[L_t] &\geq \mathbf{E}[\ell_t(v)]\\ &\geq \mathbf{E}[\ell'_t(v)]\\ &= \frac{2(1-p)}{p}\left[1 - \left(1 - \frac{p}{2}\right)^t\right].\end{aligned}$$

Using the fact that $p < 1$, we get

$$\begin{aligned}\liminf_{t\to\infty} \mathbf{E}[L_t] &\geq \frac{2(1-p)}{p}\\ &= 2\left(\frac{1}{p} - 1\right)\\ &= \frac{2e^{\lambda\left\lfloor \frac{n}{d+1}\right\rfloor}}{2 + \frac{\lambda}{1-\lambda}\left\lfloor \frac{n}{d+1}\right\rfloor} - 2.\end{aligned}$$

Thus in the limit the expected total load in the system is at least exponential in $n/d$. In particular, if the network's minimum degree is constant, the expected total load in the system is exponential in the size of the network!

**7. Conclusion.** We analyze a simple load balancing system and show that it has many desirable steady state properties under a big variety of input conditions. In particular, we derive low-degree polynomial bounds on the asymptotic expected load and waiting times of jobs in the system, and we match these expected performance results with high-probability results in a very natural restriction of the general stochastic adversary model. While there are many stability results for similar systems in the literature, our analysis of waiting time, the strength of our bounds, and our high-probability results are novel. In addition, our application of the Pemantle–Rosenthal result reveals many of the challenges in using general adversaries and will likely provide good insight for other applications. Finally, unlike much of the related work, our results hold for arbitrary connected network topologies and are optimal for the extremely important expander topology.

Plenty of open problems remain. In particular, our analysis cannot be easily modified to handle the case $\lambda = 1$, since we do not count the load that is consumed during the distribution phase of an epoch. Thus that case remains open. In addition, it is unknown whether Markovian adversaries are weaker than the general adversaries that we analyze. A result along these lines would have important ramifications in future analyses of network processes, since it would answer the question of whether Markov chain techniques can fill the role of the more general Pemantle and Rosenthal stochastic process results. And, of course, any results in an extension of our model allowing for asynchronous communication between nodes, nonuniform job execution time, and/or a continuous timescale would be very interesting.

**Acknowledgments.** We would like to thank the anonymous referees, whose elaborate comments significantly improved the overall presentation of the paper.

## REFERENCES

[1] W. Aiello, B. Awerbuch, B. M. Maggs, and S. Rao, *Approximate load balancing on dynamic and asynchronous networks*, in Proceedings of the 25th Annual ACM Symposium on Theory of Computing (STOC'93), 1993, pp. 632–641.

[2] A. Anagnostopoulos, A. Kirsch, and E. Upfal, *Stability and efficiency of a random local load balancing protocol*, in Proceedings of the 44th IEEE Symposium on Foundations of Computer Science (FOCS'03), 2003, pp. 472–481.

[3] E. Anshelevich, D. Kempe, and J. Kleinberg, *Stability of load balancing in dynamic adversarial systems*, in Proceedings of the 34th Annual ACM Symposium on Theory of Computing (STOC'02), 2002, pp. 399–406.

[4] P. Berenbrink, T. Friedetzky, and L. A. Goldberg, *The natural work-stealing algorithm is stable*, SIAM J. Comput., 32 (2003), pp. 1260–1279.

[5] A. Borodin, J. Kleinberg, P. Raghavan, M. Sudan, and D. P. Williamson, *Adversarial queuing theory*, J. ACM, 48 (2001), pp. 13–38.

[6] W. Feller, *An Introduction to Probability Theory and Its Application, Vol.* 2, 2nd ed., John Wiley and Sons, New York, 1971.

[7] B. Ghosh, F. T. Leighton, B. M. Maggs, S. Muthukrishnan, C. G. Plaxton, R. Rajaraman, A. W. Richa, R. E. Tarjan, and D. Zuckerman, *Tight analyses of two local load balancing algorithms*, SIAM J. Comput., 29 (1999), pp. 29–64.

[8] B. Ghosh and S. Muthukrishnan, *Dynamic load balancing by random matchings*, J. Comput. System Sci., 53 (1996), pp. 357–370.

[9] S. Karlin and H. M. Taylor, *A First Course in Stochastic Processes*, 2nd ed., Academic Press, New York, 1975.

[10] R. Lüling and B. Monien, *A dynamic distributed load balancing algorithm with provable good performance*, in Proceedings of the 5th ACM Symposium on Parallel Algorithms and Architectures (SPAA'93), 1993, pp. 164–172.

[11] F. Meyer auf der Heide, B. Oesterdiekhoff, and R. Wanka, *Strongly adaptive token distribution*, in Proceedings of the 20th International Colloquium on Automata, Languages and Programming (ICALP'93), 1993, pp. 398–409.

[12] S. P. Meyn and R. L. Tweedie, *Markov Chains and Stochastic Stability*, Comm. Control Engrg. Ser., Springer-Verlag, London, New York, 1993.

[13] M. Mitzenmacher, *Load balancing and density dependent jump Markov processes*, in Proceedings of 37th IEEE Conference on Foundations of Computer Science (FOCS'96), 1996, pp. 213–222.

[14] M. Mitzenmacher, *Analyses of load stealing models based on differential equations*, in Proceedings of the 10th Annual ACM Symposium on Parallel Algorithms and Architectures (SPAA'98), 1998, pp. 212–221.

[15] S. Muthukrishnan and R. Rajaraman, *An adversarial model for distributed dynamic load balancing*, in Proceedings of the 10th Annual ACM Symposium on Parallel Algorithms and Architectures (SPAA'98), 1998, pp. 47–54.

[16] D. Peleg and E. Upfal, *The token distribution problem*, SIAM J. Comput., 18 (1989), pp. 229–243.

[17] R. Pemantle and J. S. Rosenthal, *Moment conditions for a sequence with negative drift to be uniformly bounded in $L^r$*, Stochastic Process. Appl., 82 (1999), pp. 143–155.

[18] Y. Rabani, A. Sinclair, and R. Wanka, *Local divergence of Markov chains and the analysis of iterative load balancing schemes*, in Proceedings of the 39th IEEE Symposium on Foundations of Computer Science (FOCS'98), 1998, pp. 694–705.

[19] L. Rudolph, M. Slivkin-Allalouf, and E. Upfal, *A simple load balancing scheme for task allocation in parallel machines*, in Proceedings of the 3rd Annual ACM Symposium on Parallel Algorithms and Architectures (SPAA'91), 1991, pp. 237–245.

[20] C.-Z. Xu and F. C. M. Lau, *Iterative dynamic load balancing in multicomputers*, J. Oper. Res. Soc., 45 (1994), pp. 786–796.

[21] W. Zhu, C. Steketee, and B. Muilwijk, *Load balancing and workstation autonomy on Amoeba*, Aust. Comput. Sci. Commun., 17 (1995), pp. 588–597.

SIAM J. COMPUT.
Vol. 34, No. 3, pp. 640–668

# A SECOND-ORDER PERCEPTRON ALGORITHM*

NICOLÒ CESA-BIANCHI[†], ALEX CONCONI[†], AND CLAUDIO GENTILE[‡]

**Abstract.** Kernel-based linear-threshold algorithms, such as support vector machines and Perceptron-like algorithms, are among the best available techniques for solving pattern classification problems. In this paper, we describe an extension of the classical Perceptron algorithm, called second-order Perceptron, and analyze its performance within the mistake bound model of on-line learning. The bound achieved by our algorithm depends on the sensitivity to second-order data information and is the best known mistake bound for (efficient) kernel-based linear-threshold classifiers to date. This mistake bound, which strictly generalizes the well-known Perceptron bound, is expressed in terms of the eigenvalues of the empirical data correlation matrix and depends on a parameter controlling the sensitivity of the algorithm to the distribution of these eigenvalues. Since the optimal setting of this parameter is not known a priori, we also analyze two variants of the second-order Perceptron algorithm: one that adaptively sets the value of the parameter in terms of the number of mistakes made so far, and one that is parameterless, based on pseudoinverses.

**Key words.** pattern classification, mistake bounds, Perceptron algorithm

**AMS subject classifications.** 68Q32, 68W40, 68T10

**DOI.** 10.1137/S0097539703432542

## 1. Introduction.

Research in linear-threshold classifiers has been recently revamped by the popularity of kernel methods [1, 12, 36], a set of mathematical tools used to efficiently represent complex nonlinear decision surfaces in terms of linear classifiers in a high-dimensional feature space defined by kernel functions. To some extent, statistical learning theories have been able to explain why kernel methods do not suffer from the "curse of dimensionality"—that is, why they exhibit a remarkable predictive power despite the fact that the kernel-induced feature space has very many (possibly infinite) dimensions. However, these statistical results are often based on quantities, like the "margin," that provide only a superficial account of the way the predictive power is affected by the geometry of the feature space.

A different approach to the analysis of linear-threshold classifiers is the mistake bound model of on-line learning [28]. In this model, similarly to the framework of competitive analysis, the learning algorithm must be able to sequentially classify each sequence of data points making a number of mistakes not much bigger than those made by the best fixed linear-threshold classifier in hindsight (the reference predictor). The power of this approach resides in the fact that the excess loss of the learning algorithm with respect to the reference predictor can be nicely bounded in terms of the geometrical properties of any individual sequence on which the algorithm is run. Furthermore, as shown in [9], mistake bounds have corresponding statistical risk bounds that are not worse, and sometimes better, than those obtainable with a direct statistical approach.

*Received by the editors July 31, 2003; accepted for publication (in revised form) September 21, 2004; published electronically March 17, 2005. A preliminary version of this paper appeared in *Proceedings of the 15th Annual Conference on Computational Learning Theory (COLT'02)*, pp. 121–137. This research was partially supported by the European Commission under KerMIT project IST-2001-25431.

http://www.siam.org/journals/sicomp/34-3/43254.html

[†]Department of Information Sciences, Università di Milano, Italy (cesa-bianchi@dsi.unimi.it, conconi@dti.unimi.it).

[‡]DICOM, Università dell'Insubria, Varese, Italy (gentile@dsi.unimi.it).

So far, the best known mistake bound for kernel-based linear-threshold classifiers was essentially the one achieved by the classical[1] Perceptron algorithm [7, 32, 34]. In this paper we introduce an extension of the standard Perceptron algorithm, called *second-order Perceptron*.

The standard Perceptron algorithm is a popular greedy method for learning linear-threshold classifiers. Since the early sixties, it has been known that the performance of the Perceptron algorithm is governed by simple geometrical properties of the input data. As the Perceptron algorithm is essentially a gradient descent (first-order) method, we improve on it by introducing sensitivity to second-order data information. Our second-order algorithm combines the Perceptron's gradient with a sparse (and incrementally computed) version of the data correlation matrix. Our analysis shows that the second-order Perceptron algorithm is able to exploit certain geometrical properties of the data which are missed by the first-order algorithm. In particular, our bounds for the second-order algorithm depend on the distribution of the eigenvalues of the correlation matrix for the observed data sequence, as opposed to the bound for the first-order algorithm, relying only on trace information. A typical situation where the second-order bound is substantially smaller than the first-order bound is when the data lie on a flat ellipsoid, so that the eigenvalues of the correlation matrix have sharply different magnitudes, whereas the trace is dominated by the largest one.

In its basic form, the second-order Perceptron algorithm is parameterized by a constant $a > 0$, which rules the extent to which the algorithm adapts to the "warpedness" of the data. In the limit as $a$ goes to infinity our algorithm becomes the first-order Perceptron algorithm and, in that limit, the second-order bound reduces to the first-order one. The value of $a$ affects performance in a significant way. The best choice of $a$ depends on information about the learning task that is typically not available ahead of time. We develop two variants of our algorithm and prove corresponding mistake bounds. The first variant is an adaptive parameter version, while the second variant eliminates parameter $a$ and replaces standard matrix inversion with pseudo-inversion. Again, the bounds we prove are able to capture the spectral properties of the data.

In the next section, we take the classical Perceptron algorithm as a starting point for defining and motivating our second-order extension. Our description steps through an intermediate algorithm, called the *whitened Perceptron* algorithm, which serves as an illustration of the kind of spectral behavior we mean. The section is concluded with a precise definition of the learning model and with a description of the basic notation used throughout the paper.

## 2. Perceptron and whitened Perceptron.

The classical Perceptron algorithm processes a stream of examples $(\boldsymbol{x}_t, y_t)$ one at a time in trials. In trial $t$ the algorithm receives an instance vector $\boldsymbol{x}_t \in \mathbb{R}^n$ and predicts the value of the unknown label $y_t \in \{-1, +1\}$ associated with $\boldsymbol{x}_t$. The algorithm keeps a weight vector $\boldsymbol{w}_t \in \mathbb{R}^n$ representing its internal state, and its prediction at time $t$ is given by[2]

[1]Several kernel-based linear-threshold algorithms have been proposed and analyzed in recent years, including relaxation methods [14], ROMMA [27], Alma [17], and variants thereof. All these on-line algorithms are Perceptron-like, and their mistake bounds coincide with the one achieved by the standard Perceptron algorithm.

[2]Here and throughout, superscript $^\top$ denotes transposition. Also, SGN denotes the signum function $\text{SGN}(z) = 1$ if $z \ge 0$ and $\text{SGN}(z) = -1$ otherwise (we conventionally give a positive sign to zero).

$\widehat{y}_t = \textsc{sgn}(\boldsymbol{w}_t^\top \boldsymbol{x}_t) \in \{-1, 1\}$. We say that the algorithm has made a *mistake* at trial $t$ if the prediction $\widehat{y}_t$ and the label $y_t$ disagree. In such a case the algorithm updates its internal state according to the simple additive rule $\boldsymbol{w}_{t+1} = \boldsymbol{w}_t + y_t\,\boldsymbol{x}_t$, i.e., by moving the old weight vector along the direction of the instance $\boldsymbol{x}_t$ on which the algorithm turned out to be wrong (with the "right" orientation determined by $y_t = -\widehat{y}_t$). If $\widehat{y}_t = y_t$, no weight update is made (thus the number of mistakes is always equal to the number of weight updates).

A sequence of examples is linearly separable if the instance vectors $\boldsymbol{x}_t$ are consistently labeled according to whether they lie on the positive ($y_t = +1$) or the negative ($y_t = -1$) side of an unknown target hyperplane with normal vector $\boldsymbol{u} \in \mathbb{R}^n$ and passing through the origin. The well-known Perceptron convergence theorem [7, 32] states that the Perceptron algorithm is guaranteed to make at most $(R/\gamma)^2$ mistakes on any number $t$ of examples in a linearly separable sequence, where

$$\begin{aligned} R &= \max_{1\le s\le t} \|\boldsymbol{x}_s\|, \\ \gamma &= \min_{1\le s\le t} \left|\boldsymbol{u}^\top \boldsymbol{x}_t\right|. \end{aligned}$$

This implies that cycling through any sequence of linearly separable examples will eventually lead the Perceptron algorithm to compute a weight vector $\boldsymbol{w} \in \mathbb{R}^n$ classifying all examples correctly, i.e., such that $y_t = \textsc{sgn}(\boldsymbol{w}^\top \boldsymbol{x}_t)$ for all $t$ in the sequence. The convergence speed is thus critically influenced by the degree of linear separability of the examples, as expressed by the ratio $(R/\gamma)^2$. Hence, those vectors $\boldsymbol{x}_t$ which have both a small projection component over $\boldsymbol{u}$ (i.e., they are "almost" orthogonal to $\boldsymbol{u}$) and have a large norm $\|\boldsymbol{x}_t\|$ are the hardest ones for the Perceptron algorithm. The situation is illustrated in Figure 2.1. Consider the case when the algorithm observes instance vectors $\boldsymbol{x}_t$ lying in the dotted circles on opposite sides of that figure. Such vectors have small components along the direction of $\boldsymbol{u}$; hence they are intuitively irrelevant to the target vector. Still, such $\boldsymbol{x}_t$'s might significantly mislead the Perceptron algorithm (thereby slowing down its convergence). Trial $t$ depicted in Figure 2.1 is a mistaken trial for the algorithm, since $\textsc{sgn}(\boldsymbol{w}_t^\top \boldsymbol{x}_t) \neq \textsc{sgn}(\boldsymbol{u}^\top \boldsymbol{x}_t) = y_t = -1$. Now, the new weight vector $\boldsymbol{w}_{t+1}$ computed by the algorithm has a larger projection[3] onto $\boldsymbol{u}$ than the old vector $\boldsymbol{w}_t$. But the algorithm's prediction is based only upon the direction of its current weight vector. Hence the step $\boldsymbol{w}_t \to \boldsymbol{w}_{t+1}$ does not seem to make satisfactory progress toward $\boldsymbol{u}$.

Let us now turn to an algorithm which is related to both the first-order and the second-order Perceptron algorithm (described in section 3). We call this algorithm the *whitened Perceptron* algorithm. Strictly speaking, this is not an incremental algorithm. In fact, it assumes that all the instance vectors $\boldsymbol{x}_1, \boldsymbol{x}_2, \dots, \boldsymbol{x}_T \in \mathbb{R}^n$ are preliminarily available, and only the labels $y_1, y_2, \dots, y_T$ are hidden. For the sake of simplicity, assume that these vectors span $\mathbb{R}^n$. Therefore $T \ge n$, the correlation matrix $M = \sum_{t=1}^T \boldsymbol{x}_t\,\boldsymbol{x}_t^\top$ is full-rank, and $M^{-1}$ exists. Also, since $M$ is positive definite, $M^{-1/2}$ exists as well (see, e.g., [31, Chap. 3]). The whitened Perceptron algorithm is simply the standard Perceptron algorithm run on the transformed (whitened) sequence $(M^{-1/2}\boldsymbol{x}_1, y_1), (M^{-1/2}\boldsymbol{x}_2, y_2), \dots, (M^{-1/2}\boldsymbol{x}_T, y_T)$. The transformation $M^{-1/2}$ is called the whitening transform (see, e.g., [14]) and has the

[3]In its simplest form, the Perceptron convergence theorem exploits the measure of progress $\boldsymbol{u}^\top \boldsymbol{w}_t$ and is based on the fact that under linear separability assumptions this measure steadily increases through mistaken trials.

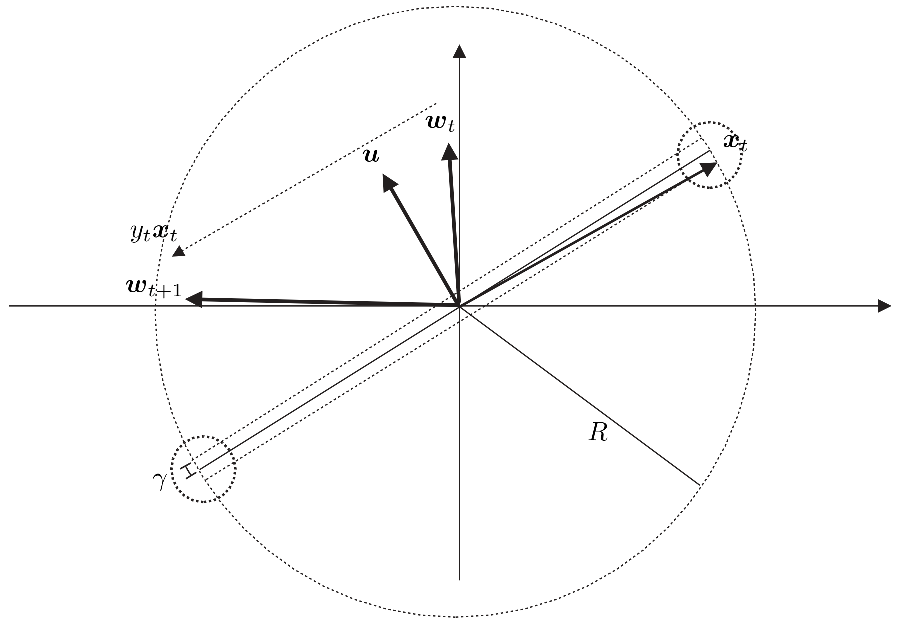


FIG. 2.1. *Behavior of the Perceptron algorithm on extreme (though linearly separable) cases. Here $\boldsymbol{u}$ denotes the hidden target vector, $\boldsymbol{w}_t$ is the weight vector maintained by the algorithm at the beginning of trial $t$, and $\boldsymbol{x}_t$ is the instance vector observed in that trial. We are assuming that all instances have bounded (Euclidean) length $R$ and that the examples are linearly separable with margin $\gamma > 0$ (so that no vector in the sequence of examples can lie within the two dotted lines running parallel to the decision boundary of $\boldsymbol{u}$). Since the angle between $\boldsymbol{u}$ and $\boldsymbol{x}_t$ is (slightly) larger than 90 degrees, the label $y_t$ is assigned the value $-1$. On the other hand, $\boldsymbol{w}_t^\top \boldsymbol{x}_t > 0$ holds; hence the algorithm makes a mistake. Now, $\boldsymbol{x}_t$ lies exactly on one of the two dotted lines and has maximal length $R$, but it has also a small projection along the direction of $\boldsymbol{u}$, meaning that the direction marked by $\boldsymbol{x}_t$ is (almost) irrelevant to $\boldsymbol{u}$. However, the simple additive rule of the Perceptron algorithm makes the new weight vector $\boldsymbol{w}_{t+1}$ farther from $\boldsymbol{u}$ than the old one.*

effect of reducing the correlation matrix of the transformed instances to the identity matrix $I_n$. In fact,

$$\begin{aligned}\sum_{t=1}^{T}\left(M^{-1/2}\boldsymbol{x}_t\right)\left(M^{-1/2}\boldsymbol{x}_t\right)^\top &= \sum_{t=1}^{T} M^{-1/2}\boldsymbol{x}_t\,\boldsymbol{x}_t^\top M^{-1/2}\\ &= M^{-1/2}M\,M^{-1/2}\\ &= I_n.\end{aligned}$$

Again for the sake of simplicity, suppose that the original sequence of examples is linearly separable: $\gamma = \min_t y_t\,\boldsymbol{u}^\top \boldsymbol{x}_t > 0$ for some unit norm vector $\boldsymbol{u}$. Then the whitened sequence is also linearly separable. In fact, hyperplane $\boldsymbol{z} = M^{1/2}\boldsymbol{u}$ separates the whitened sequence with margin $\gamma/\left\|M^{1/2}\boldsymbol{u}\right\|$. By the aforementioned convergence theorem, the number of mistakes made by the Perceptron algorithm on the whitened sequence is thus at most

$$(2.1)\qquad \frac{1}{\gamma^2}\left(\max_t \left\|M^{-1/2}\boldsymbol{x}_t\right\|^2\right)\left\|M^{1/2}\boldsymbol{u}\right\|^2 = \frac{1}{\gamma^2}\max_t\left(\boldsymbol{x}_t^\top M^{-1}\boldsymbol{x}_t\right)\left(\boldsymbol{u}^\top M\boldsymbol{u}\right).$$

To appreciate the potential advantage of whitening the data, note that when the instance vectors $\boldsymbol{x}_1,\dots,\boldsymbol{x}_T$ are very correlated the quadratic form $\boldsymbol{x}_t^\top M^{-1}\boldsymbol{x}_t$ tends

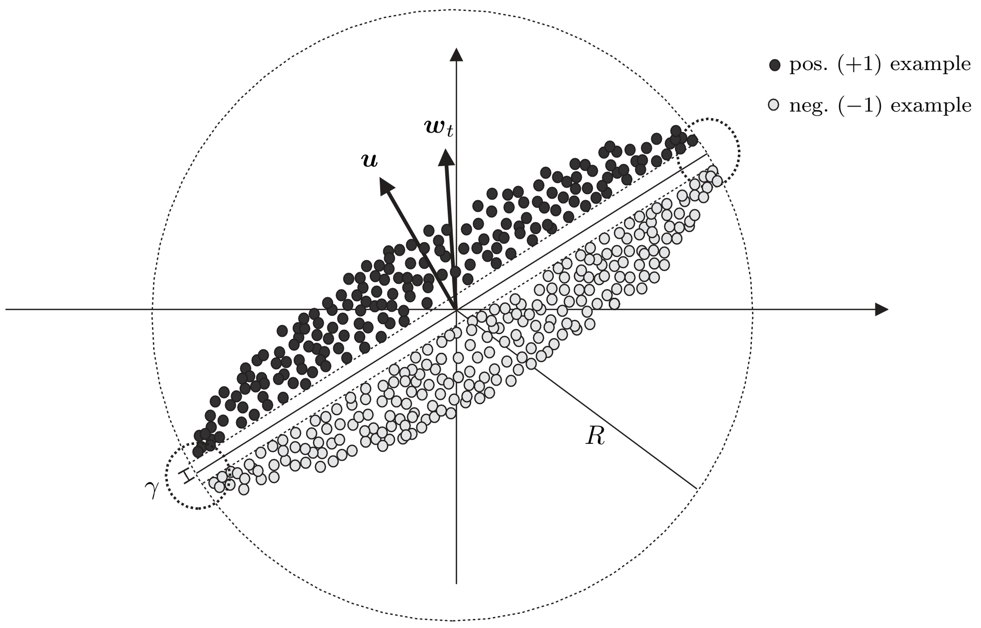


FIG. 2.2. *Scattering of data which the whitened (and the second-order) Perceptron algorithm can take advantage of. Here all instance vectors lie on a flat ellipsoid enclosed in a ball of radius $R$. The examples are linearly separable with margin $\gamma > 0$ via a hyperplane whose normal vector $\boldsymbol{u}$ is aligned with the small axis of the ellipse. Thus for any instance vector $\boldsymbol{x}_t$ the projection $|\boldsymbol{u}^\top \boldsymbol{x}_t|$ onto $\boldsymbol{u}$ is "small" (though not smaller than $\gamma$). This does not make any difference for the standard Perceptron algorithm, whose worst-case behavior is essentially ruled by the* norm *of the instances lying in the two dotted circles (recall Figure* 2.1*).*

to be quite small (the expression $\max_t \left(\boldsymbol{x}_t^\top M^{-1}\boldsymbol{x}_t\right)$ might actually be regarded as a measure of correlation of the instance vectors). Also, if the instances look like those displayed in Figure 2.2, where the separating hyperplane vector $\boldsymbol{u}$ is strongly correlated with a nondominant eigenvector of $M$ (i.e., if all instances have a small projected component onto $\boldsymbol{u}$), then the bound in the right-hand side of (2.1) can be significantly smaller than the corresponding mistake bound $\max_t \|\boldsymbol{x}_t\|^2/\gamma^2 = R^2/\gamma^2$ for the classical (nonwhitened) Perceptron algorithm.

For the sake of clarity, consider the degenerate case when all data points in Figure 2.2 are evenly spread over the two parallel lines at margin $\gamma$ (so that the ellipse sketched in that figure is maximally squashed along the direction of $\boldsymbol{u}$). It is not hard to argue that this symmetric scattering of data leads to the following eigenstructure of matrix $M$: Let $\lambda_{\min}$ and $\lambda_{\max}$ be the minimal and the maximal eigenvalues of $M$, respectively. We have that the first eigenvector of $M$ (the one associated with $\lambda_{\min}$) is aligned with $\boldsymbol{u}$, while the second eigenvector (associated with $\lambda_{\max}$) is orthogonal to $\boldsymbol{u}$ (i.e., it is parallel to the two lines mentioned above). Let us denote by $\boldsymbol{u}^\perp$ a unit norm vector orthogonal to $\boldsymbol{u}$. Now, since $\boldsymbol{u}$ has unit norm, we have

$$\lambda_{\min} = \boldsymbol{u}^\top M \boldsymbol{u} = \boldsymbol{u}^\top \left(\sum_{t=1}^{T} \boldsymbol{x}_t \boldsymbol{x}_t^\top\right) \boldsymbol{u} = \sum_{t=1}^{T} (\boldsymbol{u}^\top \boldsymbol{x}_t)^2 = \gamma^2 T. \tag{2.2}$$

Also, since $\lambda_{\min} + \lambda_{\max} = \sum_{t=1}^{T} \|\boldsymbol{x}_t\|^2$ (see, e.g., section 3.1) and since, for all $\boldsymbol{x} \in \mathbb{R}^n$,

$(\boldsymbol{u}^\top \boldsymbol{x})^2 + ((\boldsymbol{u}^\perp)^\top \boldsymbol{x})^2 = \|\boldsymbol{x}\|^2$, for each $t = 1, \dots, T$ we can write

$$\boldsymbol{x}_t M^{-1} \boldsymbol{x}_t = \frac{(\boldsymbol{u}^\top \boldsymbol{x}_t)^2}{\lambda_{\min}} + \frac{((\boldsymbol{u}^\perp)^\top \boldsymbol{x}_t)^2}{\lambda_{\max}} = \frac{\gamma^2}{\gamma^2 T} + \frac{\|\boldsymbol{x}_t\|^2 - \gamma^2}{\sum_{t=1}^T \|\boldsymbol{x}_t\|^2 - \gamma^2 T}, \tag{2.3}$$

where in the first step we used the singular value decomposition (SVD) of $M$ (see Appendix D). Hence, combining (2.2) and (2.3) as in (2.1) and simplifying, we conclude that in the extreme case we sketched, the number of mistakes made by the whitened Perceptron algorithm can be bounded by

$$\frac{1}{\gamma^2} \max_t \left(\boldsymbol{x}_t^\top M^{-1} \boldsymbol{x}_t\right) \left(\boldsymbol{u}^\top M \boldsymbol{u}\right) = 1 + \frac{R^2 T - \gamma^2 T}{\sum_{t=1}^T \|\boldsymbol{x}_t\|^2 - \gamma^2 T}.$$

Note that this bound approaches 2 as the norm of the instance vectors $\boldsymbol{x}_t$ approaches $R$ (which is just the hardest case for the standard Perceptron algorithm). Thus in this case, unlike the standard Perceptron bound $R^2/\gamma^2$, the whitened Perceptron bound tends to be a *constant*, independent of both the margin $\gamma$ and the radius $R$ of the ball containing the data.

**Learning model and notation.** In the last part of this section we precisely describe the learning model and introduce our notation (some of this notation has been used earlier). The formal model we consider is the well-known mistake bound model of incremental learning, introduced by Littlestone [28] (see also [2]) and further investigated by many authors (see, e.g., [4, 10, 11, 20, 21, 25, 26, 29, 30, 39] and the references therein).

In incremental or sequential models, learning proceeds in trials. In each trial $t = 1, 2, \dots$, the algorithm observes an *instance* vector $\boldsymbol{x}_t \in \mathbb{R}^n$ (all vectors here are understood to be column vectors) and then guesses a binary label $\widehat{y}_t \in \{-1, 1\}$. Before seeing the next vector $\boldsymbol{x}_{t+1}$, the true label $y_t \in \{-1, 1\}$ associated with $\boldsymbol{x}_t$ is revealed and the algorithm knows whether its guess $\widehat{y}_t$ for $y_t$ was correct or not. In the latter case we say that the algorithm has made a *mistake*, and we call $t$ a *mistaken trial*. Each pair $(\boldsymbol{x}_t, y_t)$ we call an *example*, and a *sequence of examples* is any sequence $\mathcal{S} = ((\boldsymbol{x}_1, y_1), (\boldsymbol{x}_2, y_2), \dots, (\boldsymbol{x}_T, y_T))$. No assumptions are made on the mechanism generating the sequence of examples. Similarly to competitive analyses of on-line algorithms [8], the goal of the learning algorithm is to minimize the amount by which its total number of mistakes on an arbitrary sequence $\mathcal{S}$ exceeds some measure of the performance of a fixed classifier (in a given *comparison class*) on the same sequence $\mathcal{S}$.

The comparison class we consider here is the set of linear-threshold classifiers parametrized by the unit norm vectors $\{\boldsymbol{u} \in \mathbb{R}^n : \|\boldsymbol{u}\| = 1\}$. Thus we speak of the linear-threshold classifier $\boldsymbol{u}$ to mean the classifier $h(\boldsymbol{x}) = \textsc{sgn}(\boldsymbol{u}^\top \boldsymbol{x})$. For technical reasons, the number of mistakes of the learning algorithm will be compared to the cumulative *hinge loss* (see, e.g., [19]) of the best linear-threshold classifier in the comparison class. For any $\gamma > 0$, the hinge loss of the linear-threshold classifier $\boldsymbol{u}$ on example $(\boldsymbol{x}, y)$ at margin $\gamma$ is $D_\gamma(\boldsymbol{u}; (\boldsymbol{x}, y)) = \max\{0, \gamma - y\,\boldsymbol{u}^\top \boldsymbol{x}\}$. Also, for a given sequence of examples $\mathcal{S} = ((\boldsymbol{x}_1, y_1), (\boldsymbol{x}_2, y_2), \dots, (\boldsymbol{x}_T, y_T))$, let $D_\gamma(\boldsymbol{u}; \mathcal{S}) = \sum_{t=1}^T D_\gamma(\boldsymbol{u}; (\boldsymbol{x}_t, y_t))$. Note that $D_\gamma(\boldsymbol{u}; \mathcal{S})/\gamma$ is a strict upper bound on the number of mistakes made by $\boldsymbol{u}$ on the same sequence $\mathcal{S}$. Moreover, if $\boldsymbol{u}$ linearly separates $\mathcal{S}$ with margin $\gamma$, i.e., if $y_t\,\boldsymbol{u}^\top \boldsymbol{x}_t \geq \gamma$ for $t = 1, \dots, T$, then $D_\gamma(\boldsymbol{u}; \mathcal{S}) = 0$.

We will prove bounds on the number of mistakes having the general form

$$\texttt{number of mistakes on } \mathcal{S} \quad \leq \quad \inf_{\gamma > 0} \inf_{\boldsymbol{u}} \left( \frac{D_\gamma(\boldsymbol{u}; \mathcal{S})}{\gamma} + \frac{\texttt{spectral}(\boldsymbol{u}; \mathcal{S})}{\gamma} \right),$$

where $\mathcal{S}$ is any sequence of examples and $\texttt{spectral}(\boldsymbol{u}; \mathcal{S})$ measures certain spectral properties arising from the interaction between $\boldsymbol{u}$ and $\mathcal{S}$. The mistake bound shown above reveals how the algorithm is able to optimally trade off between the terms $D_\gamma(\boldsymbol{u}; \mathcal{S})$ and $\texttt{spectral}(\boldsymbol{u}; \mathcal{S})$ using a *single* pass over an *arbitrary* data sequence. This aggressive adaptation can be successfully exploited in settings different from the mistake bound model. In fact, as mentioned in the conclusions, the linear-threshold classifiers generated by the second-order Perceptron algorithm can be easily shown to have a probability of mistake (risk) in the statistical model of pattern classification (see, e.g., [13]) which is tightly related to the algorithm's mistake bound. These risk bounds are actually among the best achievable by any algorithm for learning linear-threshold classifiers.

The rest of the paper is organized as follows. The next section describes and analyzes the basic form of the second-order Perceptron algorithm and also shows how to formulate the algorithm in dual form, thereby allowing the use of kernel functions. In section 4 we analyze two variants of the basic algorithm: the first variant has an adaptive parameter, and the second variant computes the pseudoinverse of the correlation matrix instead of the standard inverse. Some toy experiments are reported in section 5. Section 6 contains conclusions, final remarks, and open problems.

## 3. Second-order Perceptron, basic form.

The second-order Perceptron algorithm might be viewed as an incremental variant of the whitened Perceptron algorithm; this means that the instances in the data sequence are not assumed to be known beforehand. Besides, the second-order Perceptron algorithm is sparse; that is, the whitening transform applied to each new incoming instance is based only on a possibly very small subset of the instances observed so far.

In its basic form, the second-order Perceptron algorithm (described in Figure 3.1) takes an input parameter $a > 0$. To compute its prediction in trial $t$ the algorithm uses an $n$-row matrix $X_{k-1}$ and an $n$-dimensional weight vector $\boldsymbol{v}_{k-1}$, where subscript $k-1$ indicates the number of times matrix $X$ and vector $\boldsymbol{v}$ have been updated in the first $t-1$ trials. Initially, the algorithm sets $X_0 = \emptyset$ (the empty matrix) and $\boldsymbol{v}_0 = \mathbf{0}$. Upon receiving the $t$th instance $\boldsymbol{x}_t \in \mathbb{R}^n$, the algorithm builds the augmented matrix $S_t = [\, X_{k-1} \; \boldsymbol{x}_t \,]$ (where $\boldsymbol{x}_t$ is intended to be the last column of $S_t$) and predicts the label $y_t$ of $\boldsymbol{x}_t$ with $\widehat{y}_t = \text{SGN}\left(\boldsymbol{v}_{k-1}^\top \left(aI_n + S_t\, S_t^\top\right)^{-1} \boldsymbol{x}_t\right)$, with $I_n$ being the $n \times n$ identity matrix (the addition of $I_n$ guarantees that the above inverse always exists). If $\widehat{y}_t \neq y_t$, then a mistake occurs and the algorithm updates[4] both $\boldsymbol{v}_{k-1}$ and $X_{k-1}$. Vector $\boldsymbol{v}_{k-1}$ is updated using the Perceptron rule $\boldsymbol{v}_k = \boldsymbol{v}_{k-1} + y_t\, \boldsymbol{x}_t$, whereas matrix $X_{k-1}$ is updated using $X_k = S_t = [\, X_{k-1} \; \boldsymbol{x}_t \,]$. Note that this update implies $X_k X_k^\top = X_{k-1} X_{k-1}^\top + \boldsymbol{x}_t\, \boldsymbol{x}_t^\top$. The new matrix $X_k$ and the new vector $\boldsymbol{v}_k$ will be used in the next prediction. If $\widehat{y}_t = y_t$, no update takes place, and hence the algorithm is mistake driven [28]. Also, just after an update, matrix $X_k$ has as many columns as the number of mistakes made by the algorithm so far. Note that, unlike the Perceptron algorithm, here $\boldsymbol{w}_t$ does depend on $\boldsymbol{x}_t$ (through $S_t$). Therefore the second-order Perceptron algorithm, as described in Figure 3.1, is *not* a linear-threshold predictor.

Our algorithm might be viewed as an adaptation to on-line binary classification of the ridge regression [23] method. Indeed, our analysis is inspired by the analysis of a variant of ridge regression recently introduced by Vovk [40] and further studied by Azoury and Warmuth [5] and Forster and Warmuth [15]. This variant is an instance of Vovk's general *aggregating algorithm* and is called the "forward algorithm" in [5]. Both

[4]In the degenerate case when $\boldsymbol{x}_t = \mathbf{0}$ no update is performed.

```
**Parameter:** $a > 0$.
**Initialization:** $X_0 = \emptyset$; $\boldsymbol{v}_0 = \boldsymbol{0}$; $k = 1$.
**Repeat for** $t = 1, 2, \dots$ :
    1. get instance $\boldsymbol{x}_t \in \mathbb{R}^n$;
    2. set $S_t = [\, X_{k-1}\ \boldsymbol{x}_t \,]$;
    3. predict $\widehat{y}_t = \mathrm{SGN}(\boldsymbol{w}_t^\top \boldsymbol{x}_t) \in \{-1, +1\}$,
       where $\boldsymbol{w}_t = \left(aI_n + S_t\, S_t^\top\right)^{-1} \boldsymbol{v}_{k-1}$;
    4. get label $y_t \in \{-1, +1\}$;
    5. if $\widehat{y}_t \neq y_t$, then:
           $\boldsymbol{v}_k = \boldsymbol{v}_{k-1} + y_t\, \boldsymbol{x}_t$,
           $X_k = S_t$,
           $k \leftarrow k + 1$.
```

FIG. 3.1. *The second-order Perceptron algorithm with parameter $a > 0$.*

our algorithm and the forward algorithm predict with a weight vector $\boldsymbol{w}_t$ given by the product of the inverse correlation matrix from past trials and a linear combination of past instance vectors. In both cases the current instance $\boldsymbol{x}_t$ is incorporated into the current correlation matrix before prediction. However, unlike the forward algorithm, we only keep track of past trials where the algorithm made a mistake. To complete this qualitative analogy, we note that the analysis of our algorithm uses tools developed in [5] for the forward algorithm.

The reader might wonder whether this nonlinear dependence on the current instance is somehow necessary. As a matter of fact, if in Figure 3.1 we predicted using $X_{k-1}$ instead of the augmented matrix $S_t$, then the resulting algorithm would be a linear-threshold algorithm.[5] It is easy to show that the margin value $y_t\, \boldsymbol{w}_t^\top \boldsymbol{x}_t$ achieved by the latter algorithm is always equal to the margin value of the algorithm in Figure 3.1 multiplied by a *positive* quantity depending on all instances observed

[5]In this case, the weight vector $\boldsymbol{w}_t$ computed when the algorithm is run on a sequence of examples $((\boldsymbol{x}_1, y_1), \dots, (\boldsymbol{x}_{t-1}, y_{t-1}))$ can be defined by

$$\boldsymbol{w}_t = \operatorname*{argmin}_{\boldsymbol{v}} \left( \sum_{s \in \mathcal{M}_{t-1}} \left(\boldsymbol{v}^\top \boldsymbol{x}_s - y_s\right)^2 + a\, \|\boldsymbol{v}\|^2 \right),$$

where $\mathcal{M}_{t-1}$ is the set of all trials $s \leq t-1$ such that $y_s\, \boldsymbol{w}_s^\top \boldsymbol{x}_s < 0$, and $\boldsymbol{w}_0 = \boldsymbol{0}$. This way of writing the update rule shows that this version of the second-order Perceptron algorithm can also be viewed as a sparse variant of an on-line ridge regression algorithm, whose prediction rule is $\widehat{y}_t = \mathrm{SGN}(\boldsymbol{w}_t^\top \boldsymbol{x}_t)$, where the weight vector $\boldsymbol{w}_t$ is just

$$\boldsymbol{w}_t = \operatorname*{argmin}_{\boldsymbol{v}} \left( \sum_{s=1}^{t-1} \left(\boldsymbol{v}^\top \boldsymbol{x}_s - y_s\right)^2 + a\, \|\boldsymbol{v}\|^2 \right).$$

This is actually the approach to binary classification taken in [33], where it is called the regularized least-squares classification (RLSC) method. The reader might also want to check [37] for an approach having a similar flavor. This comparison also stresses the role played by the sparsity: First, our algorithm is more efficient than RLSC, as we have only to store a (potentially small) submatrix of the data correlation matrix—the algorithm becomes significantly more efficient on the "easy" sequences where it makes very few mistakes. Second, the mistake-driven property, which causes sparsity, is a key feature when deriving spectral bounds on the number of mistakes that hold for arbitrary data sequences: no such bounds are currently known for RLSC.

so far. Therefore, unlike the linear regression framework considered in [5, 40], for binary classification the inclusion of the current instance makes no difference: running the two algorithms on the same sequence of examples would produce the same sequence of predictions. This is true even for the second-order Perceptron algorithm with pseudoinverse described in Figure 4.2 (section 4.2).

Hence, the algorithms in Figures 3.1 and 4.2 can be equivalently viewed as generators of linear-threshold classifiers. The reason we did not formulate our algorithms directly in this way is threefold. First, the above equivalence does not hold in general when parameter $a$ changes with time, as in Figure 4.1 (section 4.1). Thus, in order to maintain a uniform style of exposition, we preferred to keep the difference between Figures 3.1, 4.2, and 4.1 as little as possible. Second, including the current instance for prediction seems to be naturally suggested by the way we analyzed the algorithms. Third, in scenarios where the margin plays a crucial role, such as the information filtering problems studied in [9], including the current instance in the margin computation seems to give a slight improvement in performance.

**3.1. Analysis.** We now claim the theoretical properties of our algorithm. The following theorem is proved in Appendix A.

THEOREM 3.1. *The number $m$ of mistakes made by the second-order Perceptron algorithm of Figure* 3.1*, run on any finite sequence $\mathcal{S} = ((\boldsymbol{x}_1, y_1), (\boldsymbol{x}_2, y_2), \dots)$ of examples, satisfies*

$$m \;\le\; \inf_{\gamma>0} \min_{\|\boldsymbol{u}\|=1} \left( \frac{D_\gamma(\boldsymbol{u};\mathcal{S})}{\gamma} + \frac{1}{\gamma}\sqrt{\left(a + \boldsymbol{u}^\top X_m\, X_m^\top \boldsymbol{u}\right) \sum_{i=1}^{n} \ln\left(1 + \lambda_i/a\right)} \right),$$

*where $\lambda_1, \dots, \lambda_n$ are the eigenvalues of $X_m\, X_m^\top$.*

Some remarks are in order.

First, observe that the quantity $\boldsymbol{u}^\top X_m\, X_m^\top \boldsymbol{u}$ in the above bound always lies between $\min_i \lambda_i$ and $\max_i \lambda_i$. In particular, $\boldsymbol{u}^\top X_m\, X_m^\top \boldsymbol{u} = \lambda_i$ when $\boldsymbol{u}$ is aligned with the eigenvector associated with $\lambda_i$. This fact entails a trade-off between the hinge loss term and the square-root term in the bound.

Second, the larger $a$ gets, the more similar the "warping" matrix $(aI_n + S_t\, S_t^\top)^{-1}$ becomes to the diagonal matrix $a^{-1} I_n$. In fact, as the reader can see from Figure 3.1, in the limit as $a \to \infty$ the second-order Perceptron algorithm becomes the classical Perceptron algorithm, and the bound in Theorem 3.1 takes the form

$$m \le \inf_{\gamma>0} \min_{\|\boldsymbol{u}\|=1} \left( \frac{D_\gamma(\boldsymbol{u};\mathcal{S})}{\gamma} + \frac{1}{\gamma}\sqrt{\sum_{i=1}^{n} \lambda_i} \right).$$

Now, since the trace of a matrix equals the sum of its eigenvalues and the nonzero eigenvalues of $X_m\, X_m^\top$ coincide with the nonzero eigenvalues of $X_m^\top\, X_m$, we can immediately see that $\sum_{i=1}^n \lambda_i = \mathrm{trace}(X_m^\top\, X_m) = \sum_{t\in\mathcal{M}} \|\boldsymbol{x}_t\|^2 \le m\left(\max_{t\in\mathcal{M}} \|\boldsymbol{x}_t\|^2\right)$, where $\mathcal{M} \subseteq \{1, 2, \dots\}$ is the set of indices of mistaken trials. Thus, setting $R^2 = \max_{t\in\mathcal{M}} \|\boldsymbol{x}_t\|^2$, we can solve the resulting bound for $m$. This gives

$$m \le \inf_{\gamma>0} \min_{\|\boldsymbol{u}\|=1} \left( \frac{R^2}{2\,\gamma^2} + \frac{D_\gamma(\boldsymbol{u};\mathcal{S})}{\gamma} + \frac{R}{\gamma}\sqrt{\frac{D_\gamma(\boldsymbol{u};\mathcal{S})}{\gamma} + \frac{R^2}{4\,\gamma^2}} \right),$$

which is the Perceptron bound in the general nonseparable case [18]. This shows that, in a certain sense, the second-order Perceptron algorithm strictly generalizes the classical Perceptron algorithm by introducing an additional parameter $a$. In general, the larger $a$ becomes, the more the algorithm in Figure 3.1 resembles the Perceptron algorithm.

Third, in the linearly separable case the mistake bound for the Perceptron algorithm is $(R/\gamma)^2$. This bound is determined by the aforementioned trace inequality $\sum_{i=1}^n \lambda_i \le R^2\, m$. The second-order algorithm, on the other hand, has the bound $\sqrt{(a + \boldsymbol{u}^\top X_m\, X_m^\top \boldsymbol{u}) \sum_{i=1}^n \ln(1+\lambda_i/a)}\,/\,\gamma$, which is determined by the core spectral quantity $\left(a + \boldsymbol{u}^\top X_m\, X_m^\top \boldsymbol{u}\right) \sum_{i=1}^n \ln(1+\lambda_i/a)$. A quick comparison of the two bounds suggests that data sequences that are linearly separable by hyperplanes whose normal vectors are nearly aligned with eigenvectors having small eigenvalues should be advantageous for the second-order Perceptron algorithm (see Figure 2.2).

A more precise quantitative comparison between the two bounds might go as follows. We first note that in order to carry out this comparison, we need to somehow renounce the second-order dependence on the eigenvalues $\lambda_i$. Introduce the notation $\lambda_{\boldsymbol{u}} = \boldsymbol{u}^\top X_m\, X_m^\top \boldsymbol{u}$. We have

$$\begin{aligned}(a+\lambda_{\boldsymbol{u}}) \sum_{i=1}^n \ln(1+\lambda_i/a) &\le \max_{\lambda_1,\dots,\lambda_n\,:\,\sum_{i=1}^n \lambda_i \le R^2 m} (a+\lambda_{\boldsymbol{u}}) \sum_{i=1}^n \ln(1+\lambda_i/a)\\ &= (a+\lambda_{\boldsymbol{u}})\; n\, \ln\left(1+\frac{R^2 m}{n\,a}\right),\end{aligned}$$

since the maximum is achieved when all $\lambda_j$ are equal to $R^2 m/n$. Now, finding conditions on the data for which the second-order algorithm is advantageous is reduced to finding conditions on $a$, $\lambda_{\boldsymbol{u}}$, and $r = R^2 m/n$ such that

$$(a+\lambda_{\boldsymbol{u}}) \ln\left(1+\frac{r}{a}\right) \le r \tag{3.1}$$

is satisfied. The next lemma, proved in Appendix B through a simple derivative argument, shows that if $\lambda_{\boldsymbol{u}} < r/2$, then $a = r\lambda_{\boldsymbol{u}}/(r - 2\lambda_{\boldsymbol{u}})$ makes (3.1) a strict inequality. With this setting of $a$, the smaller $\lambda_{\boldsymbol{u}}$ is when compared to $r/2$, the smaller the left-hand side of (3.1) is when compared to the right-hand side, that is, the smaller the second-order bound is when compared to the first-order one. Thus, as we expected, if the data tend to be "flat" and $\boldsymbol{u}$ has irrelevant components along the direction of large instance vectors (as in Figure 2.2), then it is convenient to pick a small value of $a$. In particular, when $\lambda_{\boldsymbol{u}}$ is "small" the above setting of $a$ becomes $a \simeq \lambda_{\boldsymbol{u}}$ (independent of $r$). On the other hand, the lemma also shows that whenever $\lambda_{\boldsymbol{u}} \ge r/2$, then (3.1) is satisfied (as an equality) only for $a \to \infty$. Thus when $\lambda_{\boldsymbol{u}}$ is "not small" the best thing to do is to resort to the first-order algorithm.

LEMMA 3.2. *Let $f(a,\lambda,r) = (a+\lambda)\ln\left(1+\frac{r}{a}\right)$, where $a,\lambda,r > 0$. Then the following hold:*

1. *$\lim_{a\to\infty} f(a,\lambda,r) = r$; moreover, if $\lambda$ and $r$ are such that $\lambda \ge r/2$, then $f(a,\lambda,r) \ge r$ for all $a > 0$.*
2. *If $\lambda$ and $r$ are such that $\lambda < r/2$, then setting $a = a(\lambda,r) = \frac{r\lambda}{r-2\lambda}$ gets $f(a,\lambda,r) < r$ and $\lim_{2\lambda/r\to 0} f(a,\lambda,r)/r = 0$.*

We finally observe that the running time per trial of the second-order algorithm is $\Theta(n^2)$, since by the well-known Sherman–Morrison formula (see, e.g., [24, Chap. 0]) if $\boldsymbol{x}$ is a vector and $A$ is a positive definite matrix, then so is $B = A + \boldsymbol{x}\boldsymbol{x}^\top$ and

$$B^{-1} = A^{-1} - \frac{\left(A^{-1}\boldsymbol{x}\right)\left(A^{-1}\boldsymbol{x}\right)^\top}{1+\boldsymbol{x}^\top A^{-1}\boldsymbol{x}}. \tag{3.2}$$

This formula allows us to perform matrix inversion incrementally. In particular, since the $n \times n$ positive definite matrix $aI_n + S_t\, S_t^\top$ equals $aI_n + X_{k-1}\, X_{k-1}^\top + \boldsymbol{x}_t \boldsymbol{x}_t^\top$, one can compute in time $\Theta(n^2)$ the $n$-dimensional vector $\left(aI_n + X_{k-1}\, X_{k-1}^\top\right)^{-1} \boldsymbol{x}_t$ and use it along with the Sherman–Morrison formula to obtain $\left(aI_n + S_t\, S_t^\top\right)^{-1}$, again in time $\Theta(n^2)$.

**3.2. The algorithm in dual variables and the use of kernel functions.** In this section we show that the second-order Perceptron algorithm can be equivalently formulated in dual variables. This formulation allows us to run the algorithm efficiently in any given reproducing kernel Hilbert space. As a consequence, we are able to derive a kernel version of Theorem 3.1 where the mistake bound is expressed in terms of the eigenvalues of the kernel Gram matrix.

We recall that a *kernel* function (see, e.g., [12, 36, 38]) is a nonnegative function $K : \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}$ satisfying

$$\sum_{i=1}^{m} \sum_{j=1}^{m} \alpha_i\, \alpha_j\, K(\boldsymbol{x}_i, \boldsymbol{x}_j) \ge 0$$

for all $\alpha_1, \dots, \alpha_m \in \mathbb{R}$, $\boldsymbol{x}_1, \dots, \boldsymbol{x}_m \in \mathbb{R}^n$, and $m \in \mathbb{N}$ (such functions are also called positive definite). Given a kernel $K$, we can define the linear space

$$\mathcal{V}_K = \left\{ f(\cdot) = \sum_{i=1}^{m} \alpha_i\, K(\boldsymbol{x}_i, \cdot) \,:\, \alpha_i \in \mathbb{R},\, \boldsymbol{x}_i \in \mathbb{R}^n,\, i = 1, \dots, m,\, m \in \mathbb{N} \right\},$$

with norm defined by

$$\|f\|_K = \sqrt{\sum_{i=1}^{m} \sum_{j=1}^{m} \alpha_i\, \alpha_j\, K(\boldsymbol{x}_i, \boldsymbol{x}_j)}\ .$$

If this space is completed by adding all limit points of sequences $f_1, f_2, \ldots \in \mathcal{V}_K$ that are convergent in the norm $\|f\|_K$, the resulting space, denoted by $\mathcal{H}_K$, is called the *reproducing kernel Hilbert space* (induced by the kernel $K$). Classical examples of kernel functions include the so-called polynomial kernel $K(\boldsymbol{x}, \boldsymbol{y}) = (1 + \boldsymbol{x}^\top \boldsymbol{y})^d$, where $d$ is a positive integer, and the Gaussian kernel $K(\boldsymbol{x}, \boldsymbol{y}) = \exp(-\|\boldsymbol{x} - \boldsymbol{y}\|^2/2\sigma^2)$, $\sigma > 0$.

In practice, any algorithm depending on the instance vectors $\boldsymbol{x}_i$ only through inner products $\boldsymbol{x}_i^\top \boldsymbol{x}_j$ can be turned into a more general kernel version just by replacing the standard inner products $\boldsymbol{x}_i^\top \boldsymbol{x}_j$ throughout by the kernel inner products $K(\boldsymbol{x}_i, \boldsymbol{x}_j)$.

The following theorem is a slight modification of the dual formulation of the ridge regression algorithm derived in [35].

THEOREM 3.3. *With the notation of Figure* 3.1, *let $\widetilde{\boldsymbol{y}}_t$ be the $k$-component vector whose first $k-1$ components are the labels $y_i$ where the algorithm has made a mistake up to trial $t-1$ and whose last component is* 0. *Then, for all $\boldsymbol{x}_t \in \mathbb{R}^n$, we have*

$$\boldsymbol{v}_{k-1}^\top \left(aI_n + S_t\, S_t^\top\right)^{-1} \boldsymbol{x}_t = \widetilde{\boldsymbol{y}}_t^\top \left(aI_k + G_t\right)^{-1} \left(S_t^\top\, \boldsymbol{x}_t\right),$$

*where $G_t = S_t^\top S_t$ is a $k \times k$ (Gram) matrix and $I_k$ is the $k$-dimensional identity matrix.*

*Proof.* Recalling Figure 3.1, we have $\boldsymbol{v}_{k-1} = S_t\, \widetilde{\boldsymbol{y}}_t$. This implies

$$\boldsymbol{v}_{k-1}^\top \left(aI_n + S_t\, S_t^\top\right)^{-1} = \widetilde{\boldsymbol{y}}_t^\top S_t^\top \left(aI_n + S_t\, S_t^\top\right)^{-1}.$$

Thus we need only prove that

$$S_t^\top \left(aI_n + S_t\, S_t^\top\right)^{-1} = \left(aI_k + G_t\right)^{-1} S_t^\top$$

holds. But this follows from, e.g., part (d) of Exercise 17 in [6, Chap. 1]. $\square$

From a computational standpoint, we remark that the update rule of this algorithm can exploit a known adjustment formula for partitioned matrices (see, e.g., [24, Chap. 0]). This formula relates the inverse of a matrix to the inverses of some of its submatrices. In our simple case, given a $(k-1)\times(k-1)$ positive definite matrix $A$, a $(k-1)$-dimensional column vector $\boldsymbol{b}$, and a scalar $c$, we have

$$\begin{bmatrix} A & \boldsymbol{b} \\ \boldsymbol{b}^\top & c \end{bmatrix}^{-1} = \begin{bmatrix} A^{-1} + \boldsymbol{v}\boldsymbol{v}^\top/d & -\boldsymbol{v}/d \\ -\boldsymbol{v}^\top/d & 1/d \end{bmatrix},$$

where $\boldsymbol{v} = A^{-1}\boldsymbol{b}$ and[6] $d = c - \boldsymbol{b}^\top \boldsymbol{v}$ can be computed with $\Theta(k^2)$ multiplications. Using the notation of Theorem 3.3 it is easy to see that

$$aI_k + G_t = \begin{bmatrix} aI_{k-1} + X_{k-1}^\top X_{k-1} & X_{k-1}^\top \boldsymbol{x}_t \\ \boldsymbol{x}_t^\top X_{k-1} & a + \boldsymbol{x}_t^\top \boldsymbol{x}_t \end{bmatrix}.$$

Thus, using the above formula for the inverse of partitioned matrices, it follows that the $k$-dimensional matrix $\left(aI_k + G_t\right)^{-1}$ can be computed incrementally from the $(k-1)$-dimensional matrix $\left(aI_{k-1} + X_{k-1}^\top X_{k-1}\right)^{-1}$ with only $\Theta(k^2)$ extra dot products. Also, sweeping through a sequence of $T$ examples needs only $\Theta(m^2\, T)$ dot products, where $m$ is upper bounded as in Theorem 3.1.

The following result is a kernel version of Theorem 3.1. The hinge loss of any function $f \in \mathcal{H}_K$ is defined by $D_\gamma(f;(\boldsymbol{x},y)) = \max\{0, \gamma - y\, f(\boldsymbol{x})\}$.

COROLLARY 3.4. *The number $m$ of mistakes made by the dual second-order Perceptron algorithm with kernel $K$, run on any finite sequence $\mathcal{S} = ((\boldsymbol{x}_1,y_1),(\boldsymbol{x}_2,y_2),\dots)$ of examples, satisfies*

$$m \;\le\; \inf_{\gamma>0} \min_{\|f\|_K=1} \left( \frac{D_\gamma(f;\mathcal{S})}{\gamma} + \frac{1}{\gamma}\sqrt{\left(a + \sum_{t\in\mathcal{M}} f(\boldsymbol{x}_t)^2\right) \sum_i \ln\left(1 + \lambda_i/a\right)} \right).$$

*The numbers $\lambda_i$ are the nonzero eigenvalues of the kernel Gram matrix with entries $K(\boldsymbol{x}_i,\boldsymbol{x}_j)$, where $i,j \in \mathcal{M}$ and $\mathcal{M}$ is the set of indices of mistaken trials.*

Considerations similar to those made after the statement of Theorem 3.1 apply here as well. Note that the number of nonzero eigenvalues $\lambda_i$ of the kernel Gram matrix is, in general, equal to $m$, since the very nature of kernel functions makes the dimension of the space $\mathcal{H}_K$ very large, possibly infinite (hence the kernel Gram matrix is likely to be full-rank). Note also that if a linear kernel $K(\boldsymbol{x}_i,\boldsymbol{x}_j) = \boldsymbol{x}_i^\top \boldsymbol{x}_j$ is used in Corollary 3.4, then the mistake bound of Theorem 3.1 is recovered exactly. To see this observe that $\lambda_{\boldsymbol{u}} = \boldsymbol{u}^\top X_m\, X_m^\top \boldsymbol{u} = \sum_{t\in\mathcal{M}} (\boldsymbol{u}^\top \boldsymbol{x}_t)^2$ and also that the nonzero eigenvalues of the matrix $X_m\, X_m^\top$ coincide with the nonzero eigenvalues of the Gram matrix $X_m^\top X_m$.

[6]The quantity $d$ is called the Schur complement of the augmented matrix $\left[\begin{smallmatrix} A & \boldsymbol{b} \\ \boldsymbol{b}^\top & c \end{smallmatrix}\right]$ with respect to matrix $A$. From the positive definiteness of both $A$ and $\left[\begin{smallmatrix} A & \boldsymbol{b} \\ \boldsymbol{b}^\top & c \end{smallmatrix}\right]$ it follows that $d > 0$ (see, e.g., [24, Chap. 7]).

## 4. Getting rid of parameter $a$.

A major drawback of the algorithm described in Figure 3.1 is that its input parameter $a$ is fixed ahead of time. It turns out that in any practical application the value of this parameter significantly affects performance. For instance, a bad choice of $a$ might make the second-order Perceptron algorithm similar to the first-order one, even in cases when the former should perform far better.

In this section we analyze two variants of the basic algorithm: The first variant (section 4.1) is an adaptive parameter version of the algorithm in Figure 3.1; the second variant (section 4.2) eliminates the need for the trade-off parameter $a$ by replacing the "warping" matrix $(aI_n + S_t\, S_t^\top)^{-1}$ with the matrix $(S_t\, S_t^\top)^+$, i.e., the *pseudoinverse* of $S_t\, S_t^\top$.

### 4.1. Second-order Perceptron with adaptive parameter.

In this section we refine the arguments of section 3.1 by motivating and analyzing an adaptive parameter version of the algorithm in Figure 3.1. Ideally, we would like the resulting algorithm to be able to learn on the fly the "best" $a$ for the given data sequence, such as the value of $a$ that minimizes the bound in Theorem 3.1. The optimal $a$ clearly depends on unknown quantities, like the spectral structure of data and the unknown target $\boldsymbol{u}$. This ideal choice of $a$ would be able to automatically turn the second-order algorithm into a first-order one when the actual scattering of data has, say, spherical symmetry. This is a typical eigenstructure of data which a second-order algorithm cannot take advantage of.

To make our argument, we first note that the value of $a$ in Theorem 3.1 affects the bound only through the quantity

$$(a + \lambda_{\boldsymbol{u}}) \sum_{i=1}^{n} \ln\left(1 + \lambda_i / a\right). \tag{4.1}$$

In turn, both $\lambda_{\boldsymbol{u}}$ and the $\lambda_i$'s depend only on the examples where the algorithm has made a mistake. Therefore it seems reasonable to let $a$ change only in mistaken trials (this keeps the algorithm mistake driven).

Our second-order algorithm with adaptive parameter is described in Figure 4.1. The algorithm is the same as the one in Figure 3.1, except that we now feed the algorithm with an *increasing* sequence of parameter values $\{a_k\}_{k=1,2,\ldots}$, indexed by the current number of mistakes $k$. The algorithm is analyzed in Theorem 4.1 below. From the proof of that theorem (given in Appendix C) the reader can see that any strictly increasing sequence $\{a_k\}$ results in a bound on the number of mistakes. In Theorem 4.1 we actually picked a sequence which grows *linearly* with $k$. This choice seems best according to the following simple upper bounding argument. Consider again (4.1). As we noted in section 3.1,

$$(4.1) \le (a + \lambda_{\boldsymbol{u}})\, n\, \ln\left(1 + \frac{R^2 m}{n\, a}\right), \tag{4.2}$$

where $R = \max_{t\in\mathcal{M}} \|\boldsymbol{x}_t\|$. Now, from the Cauchy–Schwarz inequality one gets $\lambda_{\boldsymbol{u}} = \sum_{t\in\mathcal{M}} (\boldsymbol{u}^\top \boldsymbol{x}_t)^2 \le R^2 m$. On the other hand, if the data sequence is linearly separable with margin $\gamma > 0$, then $\lambda_{\boldsymbol{u}} = \sum_{t\in\mathcal{M}} (\boldsymbol{u}^\top \boldsymbol{x}_t)^2 \ge \gamma^2 m$. Hence, in many interesting cases, $\lambda_{\boldsymbol{u}}$ is linear in $m$. A further glance at (4.2) allows us to conclude that, viewed as a function of $m$, the right-hand side of (4.2) cannot grow less than linearly. Moreover, this minimal growth speed is achieved when $a$ is a linear[7] function of $m$.

[7]The reader will note that if $a$ grows faster than linearly, then (4.2) is still linear in $m$. However, the resulting (multiplicative) constants in (4.2) would be larger.

```
**Parameter sequence:** $\{a_k\}_{k=1,2,\dots}$, $a_{k+1} > a_k > 0$, $k = 1, 2, \dots$.
**Initialization:** $X_0 = \emptyset$; $\boldsymbol{v}_0 = \mathbf{0}$; $k = 1$.
**Repeat for** $t = 1, 2, \dots$ :
    1. get instance $\boldsymbol{x}_t \in \mathbb{R}^n$;
    2. set $S_t = [\, X_{k-1}\ \boldsymbol{x}_t \,]$;
    3. predict $\widehat{y}_t = \mathrm{SGN}(\boldsymbol{w}_t^\top \boldsymbol{x}_t) \in \{-1, +1\}$,
       where $\boldsymbol{w}_t = \left(a_k I_n + S_t S_t^\top\right)^{-1} \boldsymbol{v}_{k-1}$;
    4. get label $y_t \in \{-1, +1\}$;
    5. if $\widehat{y}_t \neq y_t$, then
           $\boldsymbol{v}_k = \boldsymbol{v}_{k-1} + y_t \boldsymbol{x}_t$,
           $X_k = S_t$,
           $k \leftarrow k + 1$.
```

FIG. 4.1. *The second-order Perceptron algorithm with increasing parameter sequence* $\{a_k\}_{k=1,2,\dots}$.

It is important to point out that this qualitative argument does not depend on the number of nonzero eigenvalues of matrix $X_m X_m^\top$. For example, our conclusions would not change even if all instances $\boldsymbol{x}_1, \boldsymbol{x}_2, \dots$ lived in a small subspace of $\mathbb{R}^n$ or, alternatively, if we mapped the same instances into a high-dimensional feature space via a kernel function (see also the discussion after Corollary 3.4).

Theorem 4.1 below uses[8] $a_k = c\,R^2 k$, where $c$ is a small positive constant. This tuning captures the "right" order of growth of $a_k$, up to a multiplicative constant $c$, whose best choice depends again on unknown quantities.

THEOREM 4.1. *If the second-order Perceptron algorithm of Figure* 4.1 *is run with parameter sequence* $a_k = c\,R^2 k$, *where* $c > 0$, *on any finite sequence* $\mathcal{S} = ((\boldsymbol{x}_1, y_1), (\boldsymbol{x}_2, y_2), \dots)$ *of examples such that* $\|\boldsymbol{x}_t\| \le R$, *then the total number* $m$ *of mistakes satisfies*

$$m \le \inf_{\gamma>0} \min_{\|\boldsymbol{u}\|=1} \left[ \frac{D_\gamma(\boldsymbol{u};\mathcal{S})}{\gamma} + \frac{1}{\gamma}\sqrt{(a_m + \lambda_{\boldsymbol{u}}) \left( B(c,m) + \sum_{i=1}^{n} \ln\left(1 + \frac{\lambda_i}{a_m}\right)\right)} \right], \tag{4.3}$$

*where* $\lambda_1, \dots, \lambda_n$ *are the eigenvalues of* $X_m X_m^\top$, $a_m = c\,R^2 m$, *and*

$$B(c,m) = \frac{1}{c} \ln \frac{m + 1/c}{1 + 1/c}.$$

To see why the algorithm with adaptive parameter might be advantageous over the one with fixed parameter, recall the tuning argument around (3.1). There, it was shown that the second-order algorithm is able to exploit the spectral properties of data when $\lambda_{\boldsymbol{u}}$ is "small." In such a case, a good tuning of $a$ is $a = r\lambda_{\boldsymbol{u}}/(r - 2\lambda_{\boldsymbol{u}})$, which, for small values of $\lambda_{\boldsymbol{u}}$, becomes $a \simeq \lambda_{\boldsymbol{u}}$. Now, for the sake of concreteness, let us focus on the linearly separable case. We have already observed that in this case $\lambda_{\boldsymbol{u}} = c' R^2 m$, where $\gamma^2/R^2 \le c' \le 1$ and $\gamma \in (0, R]$ is the minimal margin on the data. We emphasize that the tuning $a \simeq \lambda_{\boldsymbol{u}} = c'\,R^2 m$ matches the one mentioned in Theorem 4.1 up to a scaling factor. In particular, after comparing the bounds in

[8]The presence of the scaling factor $R^2$ simplifies the analysis in Appendix C.

Theorems 3.1 and 4.1, we see that Theorem 4.1 replaces $a$ with $a_m = c\,R^2 m$, i.e., with a function having a linear dependence on $m$. The price we pay for this substitution is a further additive term $B(c,m)$ which has a mild (logarithmic) dependence on $m$. The resulting bound has an inconvenient implicit form. An upper bound on an explicit solution can be clearly computed, but the calculations get overly complicated and do not add any new insights into the heart of the matter. Therefore we feel justified in omitting any further analytical detail.

As in section 3.2, one can derive a dual formulation of the algorithm in Figure 4.1 and prove a result similar to the one contained in Corollary 3.4.

In general, the total number of mistakes made by the algorithm conveys relevant information about the specific dataset at hand. Using a parameter that scales with this number allows one to partially exploit this information. Note, for instance, that if the second-order algorithm of Figure 4.1 is making "many" mistakes (and $c$ is not too small), then the algorithm tends to behave as a first-order algorithm, since $a_k$ is growing "fast." In other words, we might view the algorithm as being able to detect that its second-order structure is not suitable to the data it is processing, thereby trying to turn itself into a first-order algorithm. On the other hand, if the algorithm is making only a few mistakes, then $a_k$ tends to remain small, meaning that the current second-order structure of the algorithm appears to be the right one for the dataset at hand. This approach to parameter tuning is similar to the *self-confident* tuning adopted in [3].

We finally note that, unlike the algorithm with fixed $a$, here incremental matrix inversion looks a bit more troublesome. In fact, making $a$ change from trial to trial results in a matrix update which is no longer a low-rank adjustment. Therefore the algorithm in Figure 4.1 (as well as its dual version) does not seem to have an update rule requiring only a quadratic number of operations per trial.

## 4.2. Second-order Perceptron with pseudoinverse.

A more radical way of dealing with the trade-off parameter $a$ is to set it to zero and replace the inverse of $aI_n + S_t S_t^\top$ with the pseudoinverse $(S_t S_t^\top)^+$. This is done in the algorithm of Figure 4.2. The matrix $(S_t S_t^\top)^+$ does always exist and coincides with $(S_t S_t^\top)^{-1}$ in the case when $S_t S_t^\top$ is nonsingular. In Appendix D we collected some relevant information about pseudoinverses and their connection to the SVD. A classical reference in which to learn about them is [6].

The following result is proved in Appendix E.

THEOREM 4.2. *If the second-order Perceptron algorithm of Figure 4.2 is run on any finite sequence $\mathcal{S} = ((\boldsymbol{x}_1, y_1), (\boldsymbol{x}_2, y_2), \dots)$ of examples such that $\|\boldsymbol{x}_t\| \le R$, then the total number $m$ of mistakes satisfies*

(4.4)
$$m \le \inf_{\gamma>0} \min_{\|\boldsymbol{u}\|=1} \left[ \frac{D_\gamma(\boldsymbol{u};\mathcal{S})}{\gamma} + \frac{1}{\gamma}\sqrt{\lambda_{\boldsymbol{u}}\left(r + \frac{1}{2}r(r+1)\,\ln\left(1 + \frac{2\,R^2}{\lambda^*}\frac{m}{r(r+1)}\right)\right)}\,\right],$$

*where $r = \operatorname{rank}(X_m X_m^\top) \le n$ and $\lambda^*$ is the minimum among the* smallest *positive eigenvalues of the matrices $X_k X_k^\top$, $k = 1, \dots, m$, produced by the algorithm during its run.*

Inequality (4.4) depends on a lower bound on the positive eigenvalues of the correlation matrices produced by the algorithm. In a sense, this dependence substitutes the dependence on $a$ in the bound of Theorem 3.1. In fact, adding the matrix $aI_n$ to

```
Initialization: $X_0 = \emptyset$; $\boldsymbol{v}_0 = \mathbf{0}$; $k = 1$.
Repeat for $t = 1, 2, \ldots$ :
    1. get instance $\boldsymbol{x}_t \in \mathbb{R}^n$;
    2. set $S_t = [\, X_{k-1}\ \boldsymbol{x}_t \,]$;
    3. predict $\widehat{y}_t = \mathrm{SGN}(\boldsymbol{w}_t^\top \boldsymbol{x}_t) \in \{-1, +1\}$,
       where $\boldsymbol{w}_t = \left(S_t\, S_t^\top\right)^+ \boldsymbol{v}_{k-1}$;
    4. get label $y_t \in \{-1, +1\}$;
    5. if $\widehat{y}_t \neq y_t$, then:
           $\boldsymbol{v}_k = \boldsymbol{v}_{k-1} + y_t\, \boldsymbol{x}_t$,
           $X_k = S_t$,
           $k \leftarrow k + 1$.
```

FIG. 4.2. *The second-order Perceptron algorithm with pseudoinverse.*

the current correlation matrix of the data might be viewed as a way of setting a lower bound on the positive eigenvalues of the resulting matrix.

As we have observed in section 4.1, when the data are linearly separable with margin $\gamma > 0$ the quantity $\lambda_{\boldsymbol{u}}$ is linear in $m$. Thus, once we set $\lambda_{\boldsymbol{u}} = c' R^2 m$, with $\gamma^2/R^2 \le c' \le 1$, and simplify, bound (4.4) takes the form

$$m \le \frac{c' R^2}{\gamma^2}\left(r + \frac{1}{2} r(r+1)\,\ln\left(1 + \frac{2\,R^2}{\lambda^*}\frac{m}{r(r+1)}\right)\right). \tag{4.5}$$

As for Theorem 4.1, the two bounds (4.4) and (4.5) are in implicit form with respect to $m$. The bounds can be made explicit at the cost of adding a few logarithmic terms. Again, we decided not to carry out such calculations since they are not very insightful.

Comparing bounds (4.4) and (4.5) to the one in Theorem 3.1, one can see that the second-order algorithm with pseudoinverse might be advantageous over the basic version when the effective dimension $r$ of the data is relatively small. Also, the bounds (4.4) and (4.5) are nonvacuous only when $r < m$.

Actually, as for the algorithm in Figure 3.1, it is not hard to turn the algorithm in Figure 4.2 into an equivalent dual form. Again, one can exploit known methods to incrementally compute the pseudoinverse (see, e.g., the so-called Greville's method in [6]), reducing the computational cost per trial from cubic to quadratic.

Unfortunately, the bounds (4.4) and (4.5) are generally not useful in the presence of kernel functions[9] since the bounds have a linear[10] dependence on $r$ which cannot be avoided in the worst case. This is due to the simple fact that each time the new instance $\boldsymbol{x}_t$ lies outside the column space of the previous matrix $X_{k-1}$ the pseudoinverse $(S_t\, S_t^\top)^+$ maps $\boldsymbol{x}_t$ to the orthogonal space of this column space, so that the algorithm has a degenerate margin $\boldsymbol{w}_t^\top \boldsymbol{x}_t = 0$.

## 5. Simulations on a toy problem.

With the purpose of empirically verifying our theoretical results, we ran our second-order Perceptron algorithm on two sets of

[9]In the kernel case, $r$ is likely to be equal to $m$, giving rise to a vacuous bound.

[10]Note that the quadratic dependence on $r$ is essentially fictitious here since the $r(r+1)$ factor occurs both at the numerator and at the denominator inside the logarithm.

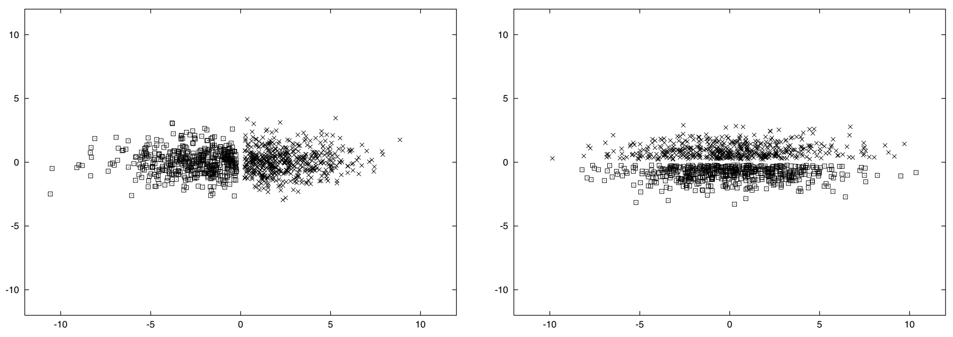


FIG. 5.1. *Projection on the two most relevant coordinates of the* 100*-dimensional datasets used in our simulations.*

TABLE 5.1

| Algorithm | Mistakes, 1st dataset | Mistakes, 2nd dataset |
|---|---|---|
| Perceptron | 30.20 (6.24) | 29.80 (8.16) |
| second-order Perceptron, $a = 1$ | 9.60 (2.94) | 5.60 (2.80) |
| second-order Perceptron, $a = 10$ | 10.60 (2.58) | 3.20 (1.47) |
| second-order Perceptron, $a = 50$ | 14.00 (4.36) | 10.40 (6.05) |

linearly separable data with 100 attributes. The two datasets are generated by sampling a Gaussian distribution whose correlation matrix has a single dominant eigenvalue, the remaining eigenvalues having the same size (in the sample realizations, the dominant eigenvalue is about eight times bigger than the others). In the first dataset (leftmost plot in Figure 5.1), labels are assigned so that the separating hyperplane is orthogonal to the eigenvector associated with the dominant eigenvalue. In the second dataset (rightmost plot in Figure 5.1), labels are assigned so that the separating hyperplane is orthogonal to the eigenvector associated with the first nondominant eigenvalue in the natural ordering of the coordinates.

According to the remarks following Theorem 3.1, we expect the Perceptron algorithm to perform similarly on both datasets (as the radius of the enclosing ball and the margin do not change between datasets), whereas the second-order Perceptron algorithm is expected to outperform the Perceptron algorithm on the second dataset. This conjecture is supported by the results in Table 5.1. These results were obtained by running the Perceptron algorithm and the second-order Perceptron algorithm (for different values of parameter $a$) for two epochs on a training set of 9000 examples, and then saving the final classifier generated by each algorithm after its last training mistake. The numbers in the table are the average number of mistakes made by these classifiers on a test set of 3000 examples, where the averages are computed over 5 random permutations of the training set (standard deviations are shown in parentheses). Normalizing the instances did not alter significantly Perceptron's performance.

To offer a familiar context for the reader experienced in empirical comparisons of learning algorithms, the predictive performance in our experiments has been evaluated using the standard test error measure. However, we could have drawn the same conclusions using, instead of the test error, the number of mistakes made by the two algorithms during training. This quantity, which is the one we bound in our theoretical results, is different from the standard training error, since in the on-line model each new mistake is made by a different classifier. A theory accounting for the relationship between the fraction of mistakes in the on-line model and the test error is developed in [9].

## 6. Conclusions and open problems.

We have introduced a second-order Perceptron algorithm, a new on-line binary classification algorithm for learning linear-threshold functions. The algorithm is able to exploit certain spectral properties of the data sequence, expressed as an interaction between the underlying target vector and the eigenvalues of the correlation matrix of the data.

The second-order Perceptron algorithm retains the standard Perceptron's key properties of sparsity and efficient dual variable representation. This allows us to efficiently run the algorithm in any reproducing kernel Hilbert space. We have proved for this algorithm the best known mistake bound for efficient kernel-based linear-threshold classifiers to date.

Since the performance of our algorithm is critically influenced by an input parameter $a$, whose optimal setting depends on the whole training set, we have developed two variants of our basic algorithm. The first variant increases the parameter $a$ as a function of the number of mistakes made. The second variant avoids altogether the parameter $a$ by replacing the inverse correlation matrix $(aI + X\,X^\top)^{-1}$ with the pseudoinverse $(X\,X^\top)^+$.

We have also run a simple experiment on synthetic data to give some evidence of the theoretical properties of our algorithm (in its basic form).

Our second-order Perceptron algorithm might be seen as a new on-line classification technique. As such, this technique could be combined with previous techniques, such as the shifting target technique [4, 22] and the approximate on-line large margin technique (see, e.g., [17, 27]).

As shown in [9], our analysis can be used to derive linear-threshold classifiers whose statistical risk can be bounded, in probability, by quantities directly related to the mistake bounds of Theorems 3.1, 4.1, and 4.2. The resulting data-dependent generalization bounds are similar in spirit, though not readily comparable, to the bounds given in [41, Thm. 5.2]. In fact, the results in [41] are derived, via involved covering numbers arguments, in terms of the correlation matrix $X\,X^\top$ of the *whole* sequence of examples. More precisely, these results are expressed in terms of all the "large" eigenvalues of $X\,X^\top$, taken in decreasing order of magnitude up to the "effective number of dimensions."[11] In contrast to that, our bounds are in terms of all the eigenvalues of the submatrix of $X\,X^\top$ made up of instances where the algorithm has made a mistake. In this sense, the sparsity of the solution produced by our algorithm is directly reflected by the magnitude of the eigenvalues of the above submatrix. To see this, go back to the primal variable form of Theorem 3.1 and observe that adding rank-one matrices of type $\boldsymbol{x}\,\boldsymbol{x}^\top$ to a correlation matrix $X\,X^\top$ results in a new correlation matrix whose eigenvalues can only be larger. Hence few mistakes are equivalent to high sparsity, which, in turn, is equivalent to small eigenvalues.

We also observe that the application of mistake bounds to the statistical learning setting is not straightforward for the whitened Perceptron algorithm described in section 2. Since the whitening matrix $M^{-1/2}$ depends on the whole training data, the whitening transformation $\boldsymbol{x} \to M^{-1/2}\boldsymbol{x}$ does not preserve stochastic independence among the (whitened) instance vectors $M^{-1/2}\boldsymbol{x}_1, \ldots, M^{-1/2}\boldsymbol{x}_T$.

There are several directions in which this work can be extended. In the following we briefly mention three of them.

[11]The "effective number of dimensions" depends, for instance, on the margin of the data.

First, it might be possible to refine the analysis of the second-order Perceptron algorithm with adaptive parameter (Theorem 4.1). The linear growth of $a_k$ is certainly a first step toward on-line parameter adaptation, though somewhat unsatisfactory since it leaves the leading coefficient of $a_k$ unspecified. Moreover, we are not aware of any incremental updating scheme for this algorithm (neither in primal nor in dual form). Second, it might be possible to refine the analysis of the second-order Perceptron with the pseudoinverse (Theorem 4.2) so as to eliminate the dependence on $\lambda^*$. Third, we would like to combine the second-order classification technology with dual-norm algorithms such as the $p$-norm algorithms [18, 20] and the Winnow/weighted majority algorithms [28, 29, 30]. This would probably give rise to new attractive algorithms for learning sparse linear-threshold functions.

**Appendix A. Proof of Theorem 3.1.** Fix an arbitrary finite sequence $\mathcal{S} = ((\boldsymbol{x}_1, y_1), (\boldsymbol{x}_2, y_2), \dots)$, and let $\mathcal{M} \subseteq \{1, 2, \dots\}$ be the set of trials where the algorithm of Figure 3.1 made a mistake. Let $A_0 = aI_n$ and $A_k = aI_n + X_k X_k^\top$. We study the evolution of $\boldsymbol{v}_k^\top A_k^{-1} \boldsymbol{v}_k$ over mistaken trials. Let $t = t_k$ be the trial where the $k$th mistake occurred. We have

$$\begin{aligned}
\boldsymbol{v}_k^\top A_k^{-1} \boldsymbol{v}_k &= (\boldsymbol{v}_{k-1} + y_t\, \boldsymbol{x}_t)^\top A_k^{-1} (\boldsymbol{v}_{k-1} + y_t\, \boldsymbol{x}_t) \\
&\qquad (\text{since } \widehat{y}_t \neq y_t \text{ implies } \boldsymbol{v}_k = \boldsymbol{v}_{k-1} + y_t\, \boldsymbol{x}_t) \\
&= \boldsymbol{v}_{k-1}^\top A_k^{-1} \boldsymbol{v}_{k-1} + 2y_t \left(A_k^{-1} \boldsymbol{v}_{k-1}\right)^\top \boldsymbol{x}_t + \boldsymbol{x}_t^\top A_k^{-1} \boldsymbol{x}_t \\
&= \boldsymbol{v}_{k-1}^\top A_k^{-1} \boldsymbol{v}_{k-1} + 2y_t\, \boldsymbol{w}_t^\top \boldsymbol{x}_t + \boldsymbol{x}_t^\top A_k^{-1} \boldsymbol{x}_t \\
&\qquad (\text{since } \widehat{y}_t \neq y_t \text{ implies } X_k = S_t \text{ and } A_k^{-1} \boldsymbol{v}_{k-1} = \boldsymbol{w}_t) \\
&\leq \boldsymbol{v}_{k-1}^\top A_k^{-1} \boldsymbol{v}_{k-1} + \boldsymbol{x}_t^\top A_k^{-1} \boldsymbol{x}_t \\
&\qquad (\text{since } \widehat{y}_t \neq y_t \text{ implies } y_t\, \boldsymbol{w}_t^\top \boldsymbol{x}_t \leq 0).
\end{aligned}$$

Now, note that $A_k$ can be recursively defined using $A_k = A_{k-1} + \boldsymbol{x}_t \boldsymbol{x}_t^\top$. Hence, applying the Sherman–Morrison formula (3.2), we get

$$\boldsymbol{v}_{k-1}^\top A_k^{-1} \boldsymbol{v}_{k-1} = \boldsymbol{v}_{k-1}^\top A_{k-1}^{-1} \boldsymbol{v}_{k-1} - \frac{\left(\boldsymbol{v}_{k-1}^\top A_{k-1}^{-1} \boldsymbol{x}_t\right)^2}{1 + \boldsymbol{x}_t^\top A_{k-1}^{-1} \boldsymbol{x}_t} \leq \boldsymbol{v}_{k-1}^\top A_{k-1}^{-1} \boldsymbol{v}_{k-1},$$

where the inequality holds since $A_{k-1}^{-1}$ is the inverse of a positive definite matrix (and so $A_{k-1}^{-1}$ is positive definite). Thus we get

$$\boldsymbol{v}_k^\top A_k^{-1} \boldsymbol{v}_k \leq \boldsymbol{v}_{k-1}^\top A_{k-1}^{-1} \boldsymbol{v}_{k-1} + \boldsymbol{x}_t^\top A_k^{-1} \boldsymbol{x}_t,$$

holding for all $k = 1, \dots, m = |\mathcal{M}|$. Summing the last inequality over $k$ (or, equivalently, over $t \in \mathcal{M}$) and using $\boldsymbol{v}_0 = \boldsymbol{0}$ yields

$$\begin{aligned}
\boldsymbol{v}_m^\top A_m^{-1} \boldsymbol{v}_m &\leq \sum_{k=1}^m \boldsymbol{x}_t^\top A_k^{-1} \boldsymbol{x}_t \\
&= \sum_{k=1}^m \left(1 - \frac{\det(A_{k-1})}{\det(A_k)}\right) \\
&\qquad (\text{using Lemma A.1 from [5] or Lemma D.1 in Appendix D}) \\
&\leq \sum_{k=1}^m \ln \frac{\det(A_k)}{\det(A_{k-1})} \qquad (\text{since } 1 - x \leq -\ln x \text{ for all } x > 0)
\end{aligned}$$

$$
\begin{aligned}
&= \ln \frac{\det(A_m)}{\det(A_0)} \\
&= \ln \frac{\det(aI_n + X_m X_m^\top)}{\det(aI_n)} \\
&= \sum_{i=1}^{n} \ln\left(1 + \frac{\lambda_i}{a}\right), \qquad \text{(A.1)}
\end{aligned}
$$

where $\lambda_1, \ldots, \lambda_n$ are the eigenvalues of $X_m\, X_m^\top$.

To conclude the proof, note that $A_m^{1/2}$ does exist since $A_m$ is positive definite (see, e.g., [31, Chap. 3]). Pick any unit norm vector $\boldsymbol{u} \in \mathbb{R}^n$ and let $\boldsymbol{z} = A_m^{1/2}\boldsymbol{u}$. Then, using the Cauchy–Schwarz inequality, we have

$$
\begin{aligned}
\sqrt{\boldsymbol{v}_m^\top A_m^{-1} \boldsymbol{v}_m} &= \left\| A_m^{-1/2} \boldsymbol{v}_m \right\| \\
&\ge \frac{\left(A_m^{-1/2}\boldsymbol{v}_m\right)^\top \boldsymbol{z}}{\|\boldsymbol{z}\|} \\
&= \frac{\boldsymbol{v}_m^\top \boldsymbol{u}}{\sqrt{\boldsymbol{u}^\top A_m \boldsymbol{u}}} \\
&= \frac{\boldsymbol{v}_m^\top \boldsymbol{u}}{\sqrt{a + \boldsymbol{u}^\top X_m\, X_m^\top \boldsymbol{u}}} \\
&\ge \frac{\gamma\, m - D_\gamma(\boldsymbol{u}; \mathcal{S})}{\sqrt{a + \boldsymbol{u}^\top X_m\, X_m^\top \boldsymbol{u}}}, \qquad \text{(A.2)}
\end{aligned}
$$

where the last inequality follows from the very definition of $D_\gamma(\boldsymbol{u}; \mathcal{S})$ and holds for any $\gamma > 0$. Putting together (A.1) and (A.2) and solving for $m$ gives the statement of the theorem.

**Appendix B. Proof of Lemma 3.2.** We first check the limiting behavior of $f$ as a function of $a$: we have $f(a, \lambda, r) \to \infty$ as $a \to 0$ and $f(a, \lambda, r) \to r$ as $a \to \infty$ for any fixed $\lambda, r > 0$. To prove the remainder of part 1, note that since $\partial^2 f / \partial a^2$ has the same sign as

$$
2\, a\, \lambda + r\, \lambda - a\, r, \qquad \text{(B.1)}
$$

it follows that when $\lambda \ge r/2$, no choice of $a > 0$ exists, which makes (B.1) negative. In this case, $f(a, \lambda, r)$, viewed as a function of $a$, is convex and decreasing. Hence its minimal value $r$ is achieved only asymptotically ($a \to \infty$).

When $\lambda < r/2$ a finite value of $a$ exists which minimizes $f$. However, computing the minimizing $a$ analytically does not seem to be easy. Therefore we resort to the approximation $a = \frac{r\lambda}{r - 2\lambda}$, which is actually the value of $a$ where (B.1) vanishes, i.e., where $f$ changes concavity. To prove part 2 we set $\beta = r/(2\lambda) > 1$ and $g(\beta) = \frac{2\beta - 1}{2\beta(\beta - 1)} \ln(2\beta - 1) = f(\frac{r\lambda}{r - 2\lambda}, \lambda, r)/r$. It thus suffices to show that $g(\beta) < 1$ for all $\beta > 1$ and $\lim_{\beta \to \infty} g(\beta) = 0$. The first statement can be proved by showing that $\lim_{\beta \to 1} g(\beta) = 1$ and that for any $\beta > 1$ the first derivative of $\ln(2\beta - 1)$ is always smaller than the first derivative of $\frac{2\beta(\beta - 1)}{2\beta - 1}$. We omit the details of these easy calculations. The second statement trivially follows from $g(\beta) = O(\ln \beta / \beta)$, as $\beta \to \infty$.

**Appendix C. Proof of Theorem 4.1.** The proof is a more refined version of the proof of Theorem 3.1 and proceeds along the same lines.

Again, we assume that the $k$th mistake occurs when processing example $(\boldsymbol{x}_t, y_t)$, $t = t_k$, and we let $\mathcal{M} \subseteq \{1, 2, \dots\}$ be the set of trials where a mistake occurred. Let $A_k = a_k I_n + X_k X_k^\top$, $k = 1, 2, \dots$, and $A_0 = a_1 I_n$. We have

$$A_k = A_{k-1} + (a_k - a_{k-1}) I_n + \boldsymbol{x}_t \boldsymbol{x}_t^\top, \quad k = 1, 2, \dots, \tag{C.1}$$

where $a_0 = a_1$. We study the evolution of the quantity $\boldsymbol{v}_k^\top A_k^{-1} \boldsymbol{v}_k$ over mistaken trials. As in the proof of Theorem 3.1, we have

$$\boldsymbol{v}_k^\top A_k^{-1} \boldsymbol{v}_k \le \boldsymbol{v}_{k-1}^\top A_k^{-1} \boldsymbol{v}_{k-1} + \boldsymbol{x}_t^\top A_k^{-1} \boldsymbol{x}_t. \tag{C.2}$$

We need to bound $\boldsymbol{v}_{k-1}^\top A_k^{-1} \boldsymbol{v}_{k-1}$ from above. From (C.1) we see that when $k \ge 2$ the matrix $A_k$ is the sum of the nonsingular matrices $A_{k-1}$ and $(a_k - a_{k-1}) I_n + \boldsymbol{x}_t \boldsymbol{x}_t^\top$ (the latter is nonsingular since $a_k > a_{k-1}$). Thus, when $k \ge 2$, computing $A_k^{-1}$ can be done through the general inversion formula

$$(B + C)^{-1} = B^{-1} - B^{-1}(B^{-1} + C^{-1})^{-1} B^{-1}$$

with $B = A_{k-1}$ and $C = (a_k - a_{k-1}) I_n + \boldsymbol{x}_t \boldsymbol{x}_t^\top$. Now, since both $B$ and $C$ are positive definite, so is the matrix $B^{-1}(B^{-1} + C^{-1})^{-1} B^{-1}$ (this easily follows from the fact that the inverse of a positive definite matrix is again positive definite and that both the sum and the product of two positive definite matrices give rise to positive definite matrices). Hence if $k \ge 2$, we can write

$$\begin{aligned}
\boldsymbol{v}_{k-1}^\top A_k^{-1} \boldsymbol{v}_{k-1} &= \boldsymbol{v}_{k-1}^\top (B + C)^{-1} \boldsymbol{v}_{k-1} \\
&= \boldsymbol{v}_{k-1}^\top \left( B^{-1} - B^{-1}(B^{-1} + C^{-1})^{-1} B^{-1} \right) \boldsymbol{v}_{k-1} \\
&\le \boldsymbol{v}_{k-1}^\top B^{-1} \boldsymbol{v}_{k-1} \\
&= \boldsymbol{v}_{k-1}^\top A_{k-1}^{-1} \boldsymbol{v}_{k-1}. \qquad \text{(C.3)}
\end{aligned}$$

On the other hand, when $k = 1$, inequality (C.3) is trivially true since $\boldsymbol{v}_0 = \boldsymbol{0}$. Thus the inequality holds for all $k \ge 1$. We plug (C.3) back into (C.2), sum over $k = 1, \dots, m = |\mathcal{M}|$, and take into account that $\boldsymbol{v}_0 = \boldsymbol{0}$. We obtain

$$\begin{aligned}
\boldsymbol{v}_m^\top A_m^{-1} \boldsymbol{v}_m &\le \sum_{k=1}^m \boldsymbol{x}_t^\top A_k^{-1} \boldsymbol{x}_t \\
&= \sum_{k=1}^m \left( 1 - \frac{\det(A_{k-1} + (a_k - a_{k-1}) I_n)}{\det(A_k)} \right) \\
&\qquad \text{(applying Lemma A.1 in [5] or Lemma D.1 in Appendix D to (C.1))} \\
&\le \sum_{k=1}^m \ln \frac{\det(A_k)}{\det(A_{k-1} + (a_k - a_{k-1}) I_n)}. \qquad \text{(C.4)}
\end{aligned}$$

The presence of matrix term $(a_k - a_{k-1}) I_n$ makes the rest of the proof a bit more involved than the proof of Theorem 3.1. Since we want to obtain bounds in terms of eigenvalues of matrices, we rewrite the rightmost side of (C.4) as a function of the eigenvalues of matrices $X_k X_k^\top$. Let $\lambda_{k,i}$ be the $i$th eigenvalue of matrix $X_k X_k^\top$, with $\lambda_{0,i} = 0$. We can write

$$\ln \frac{\det(A_k)}{\det(A_{k-1} + (a_k - a_{k-1}) I_n)} = \sum_{i=1}^n \ln \frac{a_k + \lambda_{k,i}}{a_k + \lambda_{k-1,i}}.$$

Now, simple manipulations yield

$$\begin{aligned}\sum_{k=1}^{m}\sum_{i=1}^{n}\ln\frac{a_k+\lambda_{k,i}}{a_k+\lambda_{k-1,i}} &= \sum_{k=1}^{m}\sum_{i=1}^{n}\ln\left(\frac{a_k}{a_{k-1}}\,\frac{a_{k-1}+\lambda_{k-1,i}}{a_k+\lambda_{k,i}}\right) \quad \text{(recall that } a_0=a_1\text{)}\\ &\quad+\sum_{k=1}^{m}\sum_{i=1}^{n}\ln\left(\frac{a_k+\lambda_{k,i}}{a_k+\lambda_{k-1,i}}\right)\\ &\quad+\sum_{i=1}^{n}\ln\left(\frac{a_m+\lambda_{m,i}}{a_m}\right)\\ &=\sum_{k=1}^{m}\sum_{i=1}^{n}\ln\left(\frac{a_k}{a_{k-1}}\,\frac{a_{k-1}+\lambda_{k-1,i}}{a_k+\lambda_{k-1,i}}\right)+\sum_{i=1}^{n}\ln\left(1+\frac{\lambda_{m,i}}{a_m}\right).\end{aligned}$$

Hence from (C.4) we have obtained

$$\boldsymbol{v}_m^\top A_m^{-1}\boldsymbol{v}_m \le \sum_{k=1}^{m}\sum_{i=1}^{n}\ln\left(\frac{a_k}{a_{k-1}}\,\frac{a_{k-1}+\lambda_{k-1,i}}{a_k+\lambda_{k-1,i}}\right)+\sum_{i=1}^{n}\ln\left(1+\frac{\lambda_{m,i}}{a_m}\right). \tag{C.5}$$

The reader can see that (C.5) is analogous to (A.1) but for the presence of a spurious double sum term.

We turn to upper bounding the double sum in (C.5) as a function of $m$. We proceed as follows. We first note that for $k=1$ the inner sum is zero. Therefore we continue by assuming $k\ge 2$ in the inner sum. Since $X_{k-1}\,X_{k-1}^\top$ has size $n\times n$ and has rank at most $k-1$, only $\min\{k-1,n\}$ among the eigenvalues $\lambda_{k-1,1},\dots,\lambda_{k-1,n}$ can be nonzero. Also, as we have observed elsewhere, $X_{k-1}\,X_{k-1}^\top$ has the same nonzero eigenvalues as the (Gram) matrix $X_{k-1}^\top X_{k-1}$. Since the trace of a matrix equals the sum of its eigenvalues and the trace of $X_{k-1}^\top X_{k-1}$ is at most $(k-1)\,R^2$, we have, for $k=2,\dots,m$,

$$\begin{aligned}&\sum_{i=1}^{n}\ln\left(\frac{a_k}{a_{k-1}}\,\frac{a_{k-1}+\lambda_{k-1,i}}{a_k+\lambda_{k-1,i}}\right)\\ &\le \max\left\{\sum_{j=1}^{k'}\ln\left(\frac{a_k}{a_{k-1}}\,\frac{a_{k-1}+\mu_j}{a_k+\mu_j}\right)\ :\ \mu_1,\dots,\mu_{k'}\ge 0,\ \sum_{j=1}^{k'}\mu_j\le (k-1)\,R^2\right\},\end{aligned}$$

where $k'=\min\{k-1,n\}$. The maximum is achieved when all $\mu_j$ equal $(k-1)\,R^2/k'$. Thus

$$\begin{aligned}\sum_{i=1}^{n}\ln\left(\frac{a_k}{a_{k-1}}\,\frac{a_{k-1}+\lambda_{k-1,i}}{a_k+\lambda_{k-1,i}}\right) &\le k'\,\ln\left(\frac{a_k}{a_{k-1}}\,\frac{a_{k-1}+(k-1)\,R^2/k'}{a_k+(k-1)\,R^2/k'}\right)\\ &= k'\,\ln\left(\frac{k'\,a_k\,a_{k-1}+a_k\,(k-1)\,R^2}{k'\,a_k\,a_{k-1}+a_{k-1}\,(k-1)\,R^2}\right)\end{aligned} \tag{C.6}$$

for $k=2,3,\dots,m$. Now, a derivative argument shows that the function $f(x;\alpha,\beta)=$

$x \ln\left(\frac{x+\alpha}{x+\beta}\right)$ is increasing in $x > 0$ whenever $\alpha > \beta \geq 0$. Therefore we see that

$$\begin{aligned}
\text{(C.6)} &= \frac{1}{a_k\, a_{k-1}} f\big(a_k\, a_{k-1} k';\; a_k(k-1)\, R^2,\; a_{k-1}(k-1)\, R^2\big) \\
&\leq \frac{1}{a_k\, a_{k-1}} f\big(a_k\, a_{k-1}(k-1);\; a_k(k-1)\, R^2,\; a_{k-1}(k-1)\, R^2\big) \\
&= (k-1)\, \ln\left(\frac{a_k}{a_{k-1}}\, \frac{a_{k-1}+R^2}{a_k+R^2}\right).
\end{aligned}$$

Hence we can write

$$\begin{aligned}
\sum_{k=1}^{m}\sum_{i=1}^{n} \ln\left(\frac{a_k}{a_{k-1}}\, \frac{a_{k-1}+\lambda_{k-1,i}}{a_k+\lambda_{k-1,i}}\right) &\leq \sum_{k=2}^{m}(k-1)\, \ln\left(\frac{a_k}{a_{k-1}}\, \frac{a_{k-1}+R^2}{a_k+R^2}\right) \\
&= \sum_{k=1}^{m-1} k\, \ln\left(\frac{k+1}{k}\, \frac{c\,R^2 k + R^2}{c\,R^2(k+1)+R^2}\right) \\
&\quad \text{(using } a_k = c\,R^2 k\text{)} \\
&= \sum_{k=1}^{m-1} k\, \ln\left(1 + \frac{1}{c\,k^2 + c\,k + k}\right) \\
&\leq \sum_{k=1}^{m-1} \frac{1}{c\,k + c + 1} \qquad \text{(using } \ln(1+x) \leq x\text{)} \\
&\leq \frac{1}{c}\int_{1+1/c}^{m+1/c} \frac{dx}{x} \\
&= \frac{1}{c}\, \ln\frac{m+1/c}{1+1/c} \\
&= B(c,m).
\end{aligned}$$

To bound $\sqrt{\boldsymbol{v}_m^\top A_m^{-1}\boldsymbol{v}_m}$ from below, one can proceed as in the proof of Theorem 3.1, yielding

$$\sqrt{\boldsymbol{v}_m^\top A_m^{-1}\boldsymbol{v}_m} \geq \frac{\gamma\, m - D_\gamma(\boldsymbol{u}; \mathcal{S})}{\sqrt{a_m + \boldsymbol{u}^\top X_m\, X_m^\top \boldsymbol{u}}},$$

where $a_m = c\,R^2 m$. We plug this lower bound back into (C.5) together with the previous upper bound. After rearranging we obtain

$$\left(\frac{\gamma\, m - D_\gamma(\boldsymbol{u}; \mathcal{S})}{\sqrt{a_m + \boldsymbol{u}^\top X_m\, X_m^\top \boldsymbol{u}}}\right)^2 \leq B(c,m) + \sum_{i=1}^{n} \ln\left(1 + \frac{\lambda_{m,i}}{a_m}\right).$$

Solving for $m$ occurring in the numerator of the left-hand side gives the desired bound.

**Appendix D. Pseudoinverse and SVD of a matrix.** In this section we recall basic facts about the pseudoinverse and the SVD of a matrix. This background material can be found, e.g., in [6]. We then exploit these facts to prove two technical lemmas which will be used in the subsequent sections. These lemmas build on ancillary results proven, e.g., in [5, 15, 16].

The pseudoinverse of an $n \times n$ real matrix $A$ is the (unique) $n \times n$ matrix $A^+$ satisfying the following four conditions:

$$\begin{aligned} A\,A^+A &= A, \\ A^+A\,A^+ &= A^+, \\ (A^+A)^\top &= (A^+\,A), \\ (A\,A^+)^\top &= (A\,A^+). \end{aligned}$$

If the matrix is nonsingular, then the pseudoinverse coincides with the usual inverse, i.e., $A^+ = A^{-1}$.

In order to state the properties of the pseudoinverse that we need, we exploit the well-known connection between the pseudoinverse of a matrix and its SVD. In what follows we will focus only on (symmetric) positive semidefinite matrices. This allows us to simplify the exposition and replace the notion of singular value with the more familiar notion of eigenvalue.

Let $r \le n$ be the rank of $A$. An SVD of a matrix $A$ takes the form $A = U\,D\,U^\top$, where $U$ is an orthogonal matrix (i.e., such that $U\,U^\top = U^\top U = I_n$), and $D$ is a diagonal matrix whose first $r$ diagonal entries are the positive eigenvalues $\lambda_1, \dots, \lambda_r$ of $A$, and the remaining entries are zero. The first $r$ columns of $U$ form an orthonormal basis for span$(A)$, the space spanned by the columns of $A$, whereas the remaining $n-r$ columns of $U$ form an orthonormal basis for the orthogonal complement span$^\perp(A)$. Recall that in the case under consideration, span$^\perp(A)$ coincides with null$(A)$, the null space of $A$.

A more convenient form of an SVD is one in which only the eigenvectors corresponding to positive eigenvalues are shown. Let $U_r$ be the $n \times r$ matrix made up of the first $r$ columns of $U$ and let $D_r$ be the $r \times r$ diagonal matrix whose diagonal entries are the positive eigenvalues of $A$. Then one immediately sees that the finer SVD holds:

(D.1) $$A = U_r\,D_r\,U_r^\top.$$

Every positive semidefinite matrix admits an SVD like (D.1).

Given an SVD of a matrix, computing its pseudoinverse is a rather simple matter. In particular, given (D.1), one can easily show that

(D.2) $$A^+ = U_r\,D_r^{-1}\,U_r^\top,$$

where $D_r^{-1}$ is the inverse of the (nonsingular) matrix $D_r$.

Many properties of pseudoinverses which are relevant to this paper can be derived from (D.1) and (D.2). For instance, one can immediately see that, viewed as linear transformations from $\mathbb{R}^n$ to $\mathbb{R}^n$, both $A^+A$ and $A\,A^+$ are the identical transformation onto span$(A)$ and the null transformation on span$^\perp(A)$, i.e.,

(D.3) $$A^+A\,\boldsymbol{x} = A\,A^+\boldsymbol{x} = \begin{cases} \boldsymbol{x} & \text{if } \boldsymbol{x} \in \text{span}(A), \\ 0 & \text{if } \boldsymbol{x} \in \text{span}^\perp(A). \end{cases}$$

This property easily follows from the matrix identity

(D.4) $$A^+A = A\,A^+ = U_r\,U_r^\top.$$

We are now ready to prove the following lemma, which generalizes the valuable Lemma A.1 in [5] to positive *semi*definite matrices.

Lemma D.1. *Let $A$ be an arbitrary $n \times n$ positive semidefinite matrix, let $\boldsymbol{x}$ be an arbitrary vector in $\mathbb{R}^n$, and let $B = A - \boldsymbol{x}\,\boldsymbol{x}^\top$. Then*

$$\boldsymbol{x}\,A^+\,\boldsymbol{x} = \begin{cases} 1 & \text{if } \boldsymbol{x} \notin \operatorname{span}(B), \\ 1 - \frac{\det_{\neq 0}(B)}{\det_{\neq 0}(A)} < 1 & \text{if } \boldsymbol{x} \in \operatorname{span}(B), \end{cases} \tag{D.5}$$

*where* $\det_{\neq 0}(M)$ *denotes the product of the nonzero eigenvalues of matrix $M$. Note that* $\det_{\neq 0}(M) = \det(M)$ *when $M$ is nonsingular.*

*Proof.* If $\boldsymbol{x} = 0$, the lemma is trivially verified. Hence we continue by assuming $\boldsymbol{x} \neq 0$. We first prove that $\boldsymbol{x}^\top A^+ \boldsymbol{x} = 1$ if and only if $\boldsymbol{x} \notin \operatorname{span}(B)$. From Lemma A.3 in [16] it follows that $B\,A^+\boldsymbol{x} = 0$ when $\boldsymbol{x} \notin \operatorname{span}(B)$. Hence, using (D.3) we can write

$$0 = B\,A^+\boldsymbol{x} = (A - \boldsymbol{x}\boldsymbol{x}^\top)A^+\boldsymbol{x} = \boldsymbol{x} - \boldsymbol{x}\,\boldsymbol{x}^\top A^+\boldsymbol{x},$$

which implies $\boldsymbol{x}^\top A^+ \boldsymbol{x} = 1$. On the other hand, if $\boldsymbol{x} \in \operatorname{span}(B)$, we can always apply the Sherman–Morrison formula (3.2) and conclude that

$$\boldsymbol{x}^\top A^+ \boldsymbol{x} = \boldsymbol{x}^\top (B + \boldsymbol{x}\,\boldsymbol{x}^\top)^+ \boldsymbol{x} = \frac{\boldsymbol{x}^\top B^+ \boldsymbol{x}}{1 + \boldsymbol{x}^\top B^+ \boldsymbol{x}} < 1.$$

Let us now continue by assuming $\boldsymbol{x} \in \operatorname{span}(B)$. Let $A = U_r D_r U_r^\top$ be an SVD for $A$, in the sense of (D.1). Since $U_r\,U_r^\top \boldsymbol{x} = \boldsymbol{x}$ we can write

$$B = A - \boldsymbol{x}\boldsymbol{x}^\top = U_r M U_r^\top,$$

where $M$ is the $r \times r$ symmetric matrix $M = D_r - U_r^\top \boldsymbol{x}\boldsymbol{x}^\top U_r$. We now claim that $B$ and $M$ have the same nonzero eigenvalues, so that $\det_{\neq 0}(B) = \det_{\neq 0}(M)$. Let $\mu_1, \mu_2, \ldots, \mu_k$, $k \leq r$, be the nonzero eigenvalues of $M$ and let $\boldsymbol{e}_1, \boldsymbol{e}_2, \ldots, \boldsymbol{e}_k$ be the corresponding orthonormal eigenvectors. It is then easy to verify that $\mu_1, \mu_2, \ldots, \mu_k$ are also eigenvalues of $B$ with corresponding orthonormal eigenvectors $U_r\boldsymbol{e}_1, U_r\boldsymbol{e}_2, \ldots, U_r\boldsymbol{e}_k$. Let $E_k$ be the orthonormal matrix whose columns are $\boldsymbol{e}_1, \boldsymbol{e}_2, \ldots, \boldsymbol{e}_k$, let $M = E_k S_k E_k^\top$ be an SVD of $M$, and let $\boldsymbol{v} \in \operatorname{span}^\perp(U_r E_k)$. We have

$$B\boldsymbol{v} = U_r M U_r^\top \boldsymbol{v} = U_r E_k S_k E_k^\top U_r^\top \boldsymbol{v} = 0,$$

where the last equality follows from the orthogonality $\boldsymbol{v}^\top U_r E_k = 0$. Thus $\boldsymbol{v} \in \operatorname{null}(B)$, $\operatorname{rank}(B) = \operatorname{rank}(M)$, and the only nonzero eigenvalues of $B$ are $\mu_1, \mu_2, \ldots, \mu_k$.

Computing the nonzero eigenvalues of $M$ is actually fairly straightforward. We have

$$M = D_r - U_r^\top \boldsymbol{x}\boldsymbol{x}^\top U_r = D_r(I_r - D_r^{-1} U_r^\top \boldsymbol{x}\boldsymbol{x}^\top U_r).$$

Now, $\det_{\neq 0}(D_r) = \det(D_r) = \det_{\neq 0}(A)$, while it is easy to verify by direct inspection that the matrix $I_r - D_r^{-1} U_r^\top \boldsymbol{x}\boldsymbol{x}^\top U_r$ has eigenvalue 1 with multiplicity $r-1$ corresponding to $r-1$ eigenvectors forming an orthogonal basis for $\operatorname{span}^\perp(D_r^{-1} U_r^\top \boldsymbol{x})$, and eigenvalue $1 - \boldsymbol{x}^\top U_r D_r^{-1} U_r^\top \boldsymbol{x} = 1 - \boldsymbol{x}^\top A^+ \boldsymbol{x}$ corresponding to eigenvector $D_r^{-1} U_r^\top \boldsymbol{x}$. Since $\boldsymbol{x} \in \operatorname{span}(B)$ we already know that $1 - \boldsymbol{x}^\top A^+ \boldsymbol{x} > 0$. Therefore we have obtained

$$\det_{\neq 0}(B) = \det_{\neq 0}(M) = \det(D_r) \det_{\neq 0}(I_r - D_r^{-1} U_r^\top \boldsymbol{x}\boldsymbol{x}^\top U_r) = \det_{\neq 0}(A)(1 - \boldsymbol{x}^\top A^+ \boldsymbol{x}).$$

Rearranging gives the desired result. ∎

LEMMA D.2. *Let $A$ be an arbitrary $n \times n$ positive semidefinite matrix. Set $B = A - \boldsymbol{x}\boldsymbol{x}^\top$, where $\boldsymbol{x} \notin \operatorname{span}(B)$. Then for any $\boldsymbol{v}, \boldsymbol{w} \in \operatorname{span}(B)$ we have $\boldsymbol{v}^\top A^+ \boldsymbol{w} = \boldsymbol{v}^\top B^+ \boldsymbol{w}$.*

*Proof.* We have

$$\begin{aligned}
\boldsymbol{v}^\top A^+ \boldsymbol{w} &= \boldsymbol{v}^\top B^+ B A^+ \boldsymbol{w} \\
&\qquad \text{(by (D.3) and the fact that } \boldsymbol{v} \in \operatorname{span}(B)) \\
&= \boldsymbol{v}^\top B^+ (A - \boldsymbol{x}\boldsymbol{x}^\top) A^+ \boldsymbol{w} \\
&= \boldsymbol{v}^\top B^+ A A^+ \boldsymbol{w} - \boldsymbol{v}^\top B^+ \boldsymbol{x}\boldsymbol{x}^\top A^+ \boldsymbol{w} \\
&= \boldsymbol{v}^\top B^+ \boldsymbol{w} - \boldsymbol{v}^\top B^+ \boldsymbol{x}\boldsymbol{x}^\top A^+ \boldsymbol{w} \\
&\qquad \text{(by (D.3) and the fact that } \boldsymbol{w} \in \operatorname{span}(B) \subseteq \operatorname{span}(A)) \\
&= \boldsymbol{v}^\top B^+ \boldsymbol{w} \\
&\qquad \text{(since } \boldsymbol{x}^\top A^+ \boldsymbol{w} = 0 \text{ by Lemma A.3 in [16]).} \qquad \square
\end{aligned}$$

**Appendix E. Proof of Theorem 4.2.** The proof is again similar to the proof of Theorem 3.1.

Let $\mathcal{M} \subseteq \{1, 2, \dots\}$ be the set of trials where the algorithm in Figure 4.2 made a mistake, and let $A_k = X_k X_k^\top$, $k = 1, \dots, m = |\mathcal{M}|$. We investigate how the quadratic form $\boldsymbol{v}_k^\top A_k^+ \boldsymbol{v}_k$ changes over mistaken trials. Suppose the $k$th mistake occurred in trial $t = t_k$. Proceeding as in the proof of Theorem 3.1, one can show that

$$\boldsymbol{v}_k^\top A_k^+ \boldsymbol{v}_k \le \boldsymbol{v}_{k-1}^\top A_k^+ \boldsymbol{v}_{k-1} + \boldsymbol{x}_t^\top A_k^+ \boldsymbol{x}_t. \tag{E.1}$$

Now, as in the proof of Theorem 3.1, if $\boldsymbol{x}_t \in \operatorname{span}(A_k)$, we can apply the Sherman–Morrison formula (3.2) and obtain $\boldsymbol{v}_{k-1}^\top A_k^+ \boldsymbol{v}_{k-1} \le \boldsymbol{v}_{k-1}^\top A_{k-1}^+ \boldsymbol{v}_{k-1}$. On the other hand, in the case when $\boldsymbol{x}_t \notin \operatorname{span}(A_k)$ we can apply Lemma D.2 and conclude that $\boldsymbol{v}_{k-1}^\top A_k^+ \boldsymbol{v}_{k-1} = \boldsymbol{v}_{k-1}^\top A_{k-1}^+ \boldsymbol{v}_{k-1}$. Thus in both cases $\boldsymbol{v}_{k-1}^\top A_k^+ \boldsymbol{v}_{k-1} \le \boldsymbol{v}_{k-1}^\top A_{k-1}^+ \boldsymbol{v}_{k-1}$. Combining with (E.1) we obtain

$$\boldsymbol{v}_k^\top A_k^+ \boldsymbol{v}_k \le \boldsymbol{v}_{k-1}^\top A_{k-1}^+ \boldsymbol{v}_{k-1} + \boldsymbol{x}_t^\top A_k^+ \boldsymbol{x}_t,$$

holding for all $k = 1, \dots, m$. We now sum over $k = 1, \dots, m$ using $\boldsymbol{v}_0 = \boldsymbol{0}$. We get

$$\boldsymbol{v}_m^\top A_m^+ \boldsymbol{v}_m \le \sum_{k=1}^{m} \boldsymbol{x}_t^\top A_k^+ \boldsymbol{x}_t. \tag{E.2}$$

In order to upper bound the right-hand side of (E.2), we proceed as follows. We separate the mistaken trials $k$ such that $\boldsymbol{x}_t \in \operatorname{span}(A_{k-1})$ (i.e., the trials such that $\operatorname{rank}(A_k) = \operatorname{rank}(A_{k-1})$) from the mistaken trials $k$ such that $\boldsymbol{x}_t \notin \operatorname{span}(A_{k-1})$ (i.e., the trials such that $\operatorname{rank}(A_k) = \operatorname{rank}(A_{k-1}) + 1$). Then, in applying Lemma D.1, we count $1 - \frac{\det_{\neq 0}(A_{k-1})}{\det_{\neq 0}(A_k)}$ for the first kind of trials and 1 for the second kind. Finally, we upper bound taking the worst possible case of arranging the two kinds of trials within the sequence $\mathcal{M}$, taking into account that the number of trials of the second kind is equal to $r = \operatorname{rank}(A_m)$.

We assume the following general scenario:

$$\operatorname{rank}(A_k) = \begin{cases} 1, & k = 1, 2, \dots, 1 + i_1, \\ 2, & k = 2 + i_1, 3 + i_1, \dots, 2 + i_2, \\ 3, & k = 3 + i_2, 4 + i_2, \dots, 3 + i_3, \\ \vdots & \\ r, & k = r + i_{r-1}, r + i_{r-1} + 1, \dots, r + i_r, \end{cases}$$

where $0 \le i_1 \le i_2 \le \cdots \le i_r = m - r$. With this notation the right-hand side of (E.2) can be rewritten as follows (here and in what follows we assume $i_0 = 0$):

$$\begin{aligned}
\sum_{k=1}^{m} \boldsymbol{x}_t^\top A_k^+ \boldsymbol{x}_t &= \sum_{\ell=1}^{r} \left( \boldsymbol{x}_t^\top A_{\ell+i_{\ell-1}}^+ \boldsymbol{x}_t + \sum_{k=\ell+i_{\ell-1}+1}^{\ell+i_\ell} \boldsymbol{x}_t^\top A_k^+ \boldsymbol{x}_t \right) \\
&= \sum_{\ell=1}^{r} \left( 1 + \sum_{k=\ell+i_{\ell-1}+1}^{\ell+i_\ell} \left( 1 - \frac{\det_{\neq 0}(A_{k-1})}{\det_{\neq 0}(A_k)} \right) \right) \qquad \text{(using Lemma D.1)} \\
&= r + \sum_{\ell=1}^{r} \sum_{k=\ell+i_{\ell-1}+1}^{\ell+i_\ell} \left( 1 - \frac{\det_{\neq 0}(A_{k-1})}{\det_{\neq 0}(A_k)} \right) \\
&\le r + \sum_{\ell=1}^{r} \sum_{k=\ell+i_{\ell-1}+1}^{\ell+i_\ell} \ln \frac{\det_{\neq 0}(A_k)}{\det_{\neq 0}(A_{k-1})} \\
&= r + \sum_{\ell=1}^{r} \ln \frac{\det_{\neq 0}(A_{\ell+i_\ell})}{\det_{\neq 0}(A_{\ell+i_{\ell-1}})}. \qquad \text{(E.3)}
\end{aligned}$$

We now proceed by bounding from above the logarithmic terms in (E.3). By construction we have $\operatorname{rank}(A_{\ell+i_\ell}) = \operatorname{rank}(A_{\ell+i_{\ell-1}}) = \ell$, which is also equal to the number of nonzero eigenvalues of the two matrices. Let $\lambda_i$, $i = 1, \dots, \ell$, denote the positive eigenvalues of $A_{\ell+i_{\ell-1}}$. Since $A_{\ell+i_\ell} - A_{\ell+i_{\ell-1}}$ is positive semidefinite, the positive eigenvalues of $A_{\ell+i_\ell}$ can be expressed as $\lambda_i + \mu_i$, $i = 1, \dots, \ell$, for some nonnegative values $\mu_i$, such that $\sum_{i=1}^{\ell} \mu_i \le d_\ell R^2$, where $d_\ell = i_\ell - i_{\ell-1}$ is the number of rank-one matrices of the form $\boldsymbol{x}_t \boldsymbol{x}_t^\top$ which have been added to $A_{\ell+i_{\ell-1}}$ in order to obtain $A_{\ell+i_\ell}$. We have

$$\begin{aligned}
\ln \frac{\det_{\neq 0}(A_{\ell+i_\ell})}{\det_{\neq 0}(A_{\ell+i_{\ell-1}})} &= \ln \prod_{i=1}^{\ell} \frac{\lambda_i + \mu_i}{\lambda_i} \\
&\le \sum_{i=1}^{\ell} \ln \left( 1 + \frac{\mu_i}{\lambda^*} \right) \qquad (\text{recall that } \lambda^* \le \lambda_i,\ i = 1, \dots, \ell) \\
&\le \max_{\mu_1,\dots,\mu_\ell \,:\, \sum_{i=1}^{\ell} \mu_i \le R^2 d_\ell} \sum_{i=1}^{\ell} \ln \left( 1 + \frac{\mu_i}{\lambda^*} \right) \\
&= \ell \ln \left( 1 + \frac{R^2 d_\ell}{\lambda^* \ell} \right)
\end{aligned}$$

since the maximum is achieved when $\mu_i = R^2 d_\ell / \ell$, $i = 1, \dots, \ell$. Now, since $\sum_{\ell=1}^{r} d_\ell = m - r \le m$, we can plug back into (E.3) and maximize over $d_1, \dots, d_r$ such that $\sum_{\ell=1}^{r} d_\ell \le m$. We have

$$\begin{aligned}
\sum_{t \in \mathcal{M}} \boldsymbol{x}_t^\top A_k^+ \boldsymbol{x}_t &\le r + \sum_{\ell=1}^{r} \ell \ln \left( 1 + \frac{R^2 d_\ell}{\lambda^* \ell} \right) \\
&\le r + \max_{d_1,\dots,d_r \,:\, \sum_{\ell=1}^{r} d_\ell \le m} \sum_{\ell=1}^{r} \ell \ln \left( 1 + \frac{R^2 d_\ell}{\lambda^* \ell} \right)
\end{aligned}$$

$$= r + \frac{r(r+1)}{2} \ln\left(1 + \frac{2R^2}{\lambda^*}\frac{m}{r(r+1)}\right) \tag{E.4}$$

(since the maximum is achieved when $d_\ell = \frac{2\ell m}{r(r+1)}$, $\ell = 1, \ldots, r$),

which is the required upper bound on the right-hand side of (E.2).

When $\boldsymbol{u} \notin \mathrm{span}^\perp(A_m)$ we can bound $\sqrt{\boldsymbol{v}_m^\top A_m^+ \boldsymbol{v}_m}$ from below as in the proof of Theorem 3.1. This yields

$$\sqrt{\boldsymbol{v}_m^\top A_m^+ \boldsymbol{v}_m} \ge \frac{\gamma\, m - D_\gamma(\boldsymbol{u}; \mathcal{S})}{\sqrt{\boldsymbol{u}^\top X_m\, X_m^\top \boldsymbol{u}}}.$$

Plugging back into (E.2) and combining with (E.4) gives

$$\left(\frac{\gamma\, m - D_\gamma(\boldsymbol{u}; \mathcal{S})}{\sqrt{\boldsymbol{u}^\top X_m\, X_m^\top \boldsymbol{u}}}\right)^2 \le r + \frac{r(r+1)}{2} \ln\left(1 + \frac{2R^2}{\lambda^*}\frac{m}{r(r+1)}\right).$$

Solving for $m$ occurring in the numerator of the left-hand side gives (4.4). On the other hand, when $\boldsymbol{u} \in \mathrm{span}^\perp(A_m)$ we have $\frac{D_\gamma(\boldsymbol{u};\mathcal{S})}{\gamma} = m$ and $\boldsymbol{u}^\top X_m\, X_m^\top \boldsymbol{u} = 0$, and thus (4.4) is vacuously verified as an equality.

**Acknowledgments.** Thanks to Bob Williamson for fruitful conversations and to the anonymous reviewers, whose comments turned out to be very useful in improving the presentation of this paper.

## REFERENCES

[1] M. Aizerman, E. Braverman, and L. Rozonoer, *Theoretical foundations of the potential function method in pattern recognition learning*, Autom. Remote Control, 25 (1964), pp. 821–837.

[2] D. Angluin, *Queries and concept learning*, Machine Learning, 2 (1988), pp. 319–342.

[3] P. Auer, N. Cesa-Bianchi, and C. Gentile, *Adaptive and self-confident on-line learning algorithms*, J. Comput. System Sci., 64 (2002), pp. 48–75.

[4] P. Auer and M. Warmuth, *Tracking the best disjunction*, Machine Learning, 32 (1998), pp. 127–150.

[5] K. Azoury and M. Warmuth, *Relative loss bounds for on-line density estimation with the exponential family of distributions*, Machine Learning, 43 (2001), pp. 211–246.

[6] A. Ben-Israel and T. Greville, *Generalized Inverses: Theory and Applications*, John Wiley and Sons, New York, 1974.

[7] H. Block, *The Perceptron: A model for brain functioning*, Rev. Modern Phys., 34 (1962), pp. 123–135.

[8] A. Borodin and R. El-Yaniv, *Online Computation and Competitive Analysis*, Cambridge University Press, Cambridge, UK, 1998.

[9] N. Cesa-Bianchi, A. Conconi, and C. Gentile, *On the generalization ability of on-line learning algorithms*, IEEE Trans. Inform. Theory, 50 (2004), pp. 2050–2057.

[10] N. Cesa-Bianchi, Y. Freund, D. Helmbold, D. Haussler, R. Schapire, and M. Warmuth, *How to use expert advice*, J. ACM, 44 (1997), pp. 427–485.

[11] N. Cesa-Bianchi, Y. Freund, D. Helmbold, and M. Warmuth, *On-line prediction and conversion strategies*, Machine Learning, 25 (1996), pp. 71–110.

[12] N. Cristianini and J. Shawe-Taylor, *An Introduction to Support Vector Machines*, Cambridge University Press, Cambridge, UK, 2001.

[13] L. Devroye, L. Győrfi, and G. Lugosi, *A Probabilistic Theory of Pattern Recognition*, Springer-Verlag, New York, 1996.

[14] R. Duda, P. Hart, and D. G. Stork, *Pattern Classification*, John Wiley and Sons, New York, 2000.

[15] J. Forster and M. Warmuth, *Relative expected instantaneous loss bounds*, J. Comput. System Sci., 64 (2002), pp. 76–102.

[16] J. Forster and M. Warmuth, *Relative loss bounds for temporal-difference learning*, Machine Learning, 51 (2003), pp. 23–50.

[17] C. Gentile, *A new approximate maximal margin classification algorithm*, J. Machine Learning Res., 2 (2001), pp. 213–242.

[18] C. Gentile, *The robustness of the p-norm algorithms*, Machine Learning, 53 (2003), pp. 265–299.

[19] C. Gentile and M. Warmuth, *Linear hinge loss and average margin*, in Proceedings of the 1998 Conference on Advances in Neural Information Processing Systems 10, MIT Press, Cambridge, MA, 1999, pp. 225–231.

[20] A. Grove, N. Littlestone, and D. Schuurmans, *General convergence results for linear discriminant updates*, Machine Learning, 43 (2001), pp. 173–210.

[21] M. Herbster and M. Warmuth, *Tracking the best expert*, Machine Learning, 32 (1998), pp. 151–178.

[22] M. Herbster and M. Warmuth, *Tracking the best linear predictor*, J. Machine Learning Res., 1 (2001), pp. 281–309.

[23] A. Hoerl and R. Kennard, *Ridge regression: Biased estimation for nonorthogonal problems*, Technometrics, 12 (1970), pp. 55–67.

[24] R. Horn and C. Johnson, *Matrix Analysis*, Cambridge University Press, Cambridge, UK, 1985.

[25] J. Kivinen and M. Warmuth, *Relative loss bounds for multidimensional regression problems*, Machine Learning, 45 (2001), pp. 301–329.

[26] J. Kivinen, M. Warmuth, and P. Auer, *The Perceptron algorithm vs. Winnow: Linear vs. logarithmic mistake bounds when few input variables are relevant*, Artificial Intelligence, 97 (1997), pp. 325–343.

[27] Y. Li and P. Long, *The relaxed online maximum margin algorithm*, Machine Learning, 46 (2002), pp. 361–387.

[28] N. Littlestone, *Learning quickly when irrelevant attributes abound: A new linear-threshold algorithm*, Machine Learning, 2 (1988), pp. 285–318.

[29] N. Littlestone, *Redundant noisy attributes, attribute errors, and linear threshold learning using Winnow*, in Proceedings of the 4th Annual Workshop on Computational Learning Theory, Morgan-Kaufmann, San Mateo, CA, 1991, pp. 147–156.

[30] N. Littlestone and M. Warmuth, *The weighted majority algorithm*, Inform. and Comput., 108 (1994), pp. 212–261.

[31] M. Marcus and H. Minc, *Introduction to Linear Algebra*, Dover, New York, 1965.

[32] A. Novikoff, *On convergence proofs of Perceptrons*, in Proceedings of the Symposium on the Mathematical Theory of Automata, Vol. XII, Polytechnic Institute of Brooklyn, 1962, pp. 615–622.

[33] R. Rifkin, G. Yeo, and T. Poggio, *Regularized least squares classification*, in Advances in Learning Theory: Methods, Model and Applications, NATO Sci. Ser. III Comput. Systems Sci. 190, IOS Press, Amsterdam, 2003, pp. 131–153.

[34] F. Rosenblatt, *The Perceptron: A probabilistic model for information storage and organization in the brain*, Psych. Rev., 65 (1958), pp. 386–408.

[35] G. Saunders, A. Gammerman, and V. Vovk, *Ridge regression learning algorithm in dual variables*, in Proceedings of the 15th International Conference on Machine Learning, Morgan-Kaufmann, San Francisco, CA, 1998, pp. 515–521.

[36] B. Schölkopf and A. Smola, *Learning with Kernels*, MIT Press, Cambridge, MA, 2002.

[37] J. A. K. Suykens, T. Van Gestel, J. De Brabanter, B. De Moor, and J. Vandewalle, *Least Squares Support Vector Machines*, World Scientific, Singapore, 2002.

[38] V. Vapnik, *Statistical Learning Theory*, John Wiley and Sons, New York, 1998.

[39] V. Vovk, *Aggregating strategies*, in Proceedings of the 3rd Annual Workshop on Computational Learning Theory, Morgan-Kaufmann, San Mateo, CA, 1990, pp. 372–383.

[40] V. Vovk, *Competitive on-line statistics*, Internat. Statist. Rev., 69 (2001), pp. 213–248.

[41] R. Williamson, A. Smola, and B. Schölkopf, *Generalization bounds for regularization networks and support vector machines via entropy numbers of compact operators*, IEEE Trans. Inform. Theory, 47 (2001), pp. 2516–2532.

SIAM J. Comput.
Vol. 34, No. 3, pp. 669–682

# NONMIGRATORY ONLINE DEADLINE SCHEDULING ON MULTIPROCESSORS[*]

HO-LEUNG CHAN[†], TAK-WAH LAM[†], AND KAR-KEUNG TO[†]

**Abstract.** In this paper we consider multiprocessor scheduling with hard deadlines and investigate the cost of eliminating migration in the online setting. Let $I$ be any set of jobs that can be completed by some migratory offline schedule on $m$ processors. We show that $I$ can also be completed by a nonmigratory online schedule using $m$ speed-5.828 processors (i.e., processors 5.828 times faster). This result supplements the previous results that $I$ can also be completed by a nonmigratory offline schedule using $6m$ unit-speed processors [B. Kalyanasundaram and K. R. Pruhs, *J. Algorithms*, 38 (2001), pp. 2–24] or a migratory online schedule using $m$ speed-2 processors [C. A. Phillips et al., *Algorithmica*, 32 (2002), pp. 163–200]. Our result is based on a simple conservative scheduling algorithm called PARK, which commits a processor to a job only when the processor has zero commitment before its deadline. A careful analysis of PARK further shows that the processor speed can be reduced arbitrarily close to 1 by exploiting more processors (say, using $16m$ speed-1.8 processors). PARK also finds application in overloaded systems; it gives the first online nonmigratory algorithm that can exploit moderately faster processors to match the performance of any migratory offline algorithm.

**Key words.** online algorithms, multiprocessor scheduling, competitive analysis, resource augmentation, job migration

**AMS subject classifications.** 68Q25, 68W15, 68W40

**DOI.** 10.1137/S0097539703435765

## 1. Introduction.

In this paper we consider the online hard-deadline scheduling problem on multiprocessors; our focus is on eliminating migration among the processors. The hard-deadline scheduling problem is defined as follows: There are $m$ identical processors. Given a set of jobs, each characterized by its release time, deadline, and processing time (work), the objective is to schedule all the jobs on the processors so that each job can be completed by its deadline. Note that each job can be scheduled on at most one processor at any time. Preemption is allowed and a preempted job can resume later from the point of preemption. If migration is allowed, a job can resume on a different processor. From a practical point of view, nonmigratory schedules are preferable since migration may incur significant overhead. Yet allowing migration simplifies the design of scheduling algorithms.

Let $I$ be a set of jobs that can be completed by an offline schedule that allows migration. Kalyanasundaram and Pruhs [8] showed that migration is actually of limited power in offline scheduling; given a migratory offline schedule on $m$ processors, it is always possible to construct a nonmigratory offline schedule on $6m-5$ processors.[1] On the other hand, scheduling $I$ online (i.e., the information about a job is known only when it is released) is very difficult even if migration is allowed. In fact, $I$ does

[*]Received by the editors October 3, 2003; accepted for publication September 13, 2004; published electronically March 17, 2005. A preliminary version of this paper appeared in the *Proceedings of the 15th Annual ACM-SIAM Symposium on Discrete Algorithms*, SIAM, Philadelphia, ACM, New York, 2004, pp. 963–972. This research was supported in part by Hong Kong RGC grant HKU-7024/01E.
http://www.siam.org/journals/sicomp/34-3/43576.html

[†]Department of Computer Science, University of Hong Kong, Hong Kong (hlchan@cs.hku.hk, twlam@cs.hku.hk, kkto@cs.hku.hk).

[1]The construction runs in pseudopolynomial time but can be improved to run in polynomial time if the processor bound is raised from $6m-5$ to $12m-5$.

not admit any online schedule on $m$ processors if $m \geq 2$ [5].[2] Nevertheless, Phillips et al. showed that $I$ can be completed by a migratory online schedule on $m$ processors that are two times faster [13]. Migration seems to be a necessary tool in all online algorithms that are known to be able to complete $I$ on $m$ moderately faster processors (e.g., EDF (earliest deadline first), LLF (least laxity first), FR [13, 11]).

**Main result.** In this paper we devise a conservative online algorithm to show that $I$ can be completed by a nonmigratory online schedule on $m$ speed-5.828 processors. This result supplements the result of Kalyanasundaram and Pruhs [8], as it illustrates that allowing migration in the online setting also gives limited power. A careful analysis of the new algorithm further shows that the speed requirement of 5.828 can also be reduced arbitrarily close to 1 by exploiting more processors (say, using $16m$ speed-1.8 processors). More precisely, we show that for any $\epsilon > 0$, $I$ can be completed by a nonmigratory online schedule on $\lceil (1+\frac{1}{\epsilon})^2 \rceil\, m$ speed-$(1+\epsilon)^2$ processors.

**Laxity assumption.** It is widely believed that jobs with very tight deadlines or very small laxity (i.e., deadline minus release time minus work) make online scheduling difficult. Our nonmigratory online scheduling is no exception; in this paper we show that advance knowledge of the laxity of jobs in $I$ can reduce the speed or processor requirement for a nonmigratory online schedule. For example, if the work requested for every job in $I$ is at most one-eighth of its span (i.e., deadline minus release time), then we can have a nonmigratory online schedule for $I$ on $m$ speed-2.5 processors or on $4m$ unit-speed processors. Note that the latter result does not demand the presence of faster processors. In general, if the work-span ratio is at most $w$, then $I$ can be completed by a nonmigratory online schedule on $m$ speed-$(4w+2)$ processors or $\lceil 2/(1-4w) \rceil\, m$ unit-speed processors. (The latter result holds only for $w < 1/4$.)

**PARK.** The core of our results is a simple conservative online scheduling algorithm called PARK; it does not commit a processor to any job unless the processor has zero commitment before its deadline. This algorithm often lets a job wait for a while before admitting it to a processor. The conservative nature of PARK keeps any processor from being overcrowded and being tricked by the offline adversary. Like many other online algorithms, PARK is very simple but the analysis of its performance is relatively complicated. In this paper we present two different analyses of PARK showing different dimensions of its performance.

**Firm-deadline scheduling.** PARK also finds applications in scheduling overloaded systems, in which there may be too many jobs to be completed and the deadlines are firm (instead of hard) in the sense that failing to complete a job by its deadline causes only a loss in value due to that job and does not cause a system failure. Given a set $I$ of such jobs, the objective of a scheduler is to maximize the value obtained from completing the jobs. For any $c \geq 1$, an online algorithm is said to be $c$-competitive if, for any job sequence, it can obtain at least a fraction of $1/c$ of the total value obtained by any offline schedule on $m$ speed-1 processors. Consider the special case when the value of a job is proportional to its processing time. The work of Koren and Shasha [10] gave a nonmigratory online algorithm on $m$ speed-1 processors that is 3-competitive. Lam and To [12] showed that, if migration is allowed, it is possible to devise an online algorithm on $m$ speed-3 processors that is 1-competitive. Based on the latter result, we can actually extend PARK to give a nonmigratory online algorithm on $m$ speed-10 processors that is 1-competitive.

In the context of offline scheduling, the study of the power of migration is centered on the notion $\omega_m$, which is defined as the maximum over all input $I$, the ratio of the

[2]When $m = 1$, the earliest deadline first (EDF) strategy guarantees the completion of $I$ [4].

value attained by the optimal migratory offline schedule for $I$ to the value attained by the optimal nonmigratory offline schedule for $I$ [9, 8]. For jobs with arbitrary values, the best upper bound of $\omega_m$ is $(6m-5)/m$ [8]. This paper contributes to the knowledge of $\omega_m$ for the case when jobs are assumed to have a certain laxity. More precisely, our work implies that $\omega_m \le \min(6, \lceil \frac{2}{1-4w} \rceil)$ if the work-span ratio is at most $w < 1/4$.

**Organization of the paper.** Section 2 presents the new online algorithm PARK and discusses several interesting properties of PARK. Section 3 gives the first analysis of PARK, showing how to obtain a nonmigratory online schedule on $m$ speed-5.828 processors. As a byproduct, we show how to improve the resource bounds when all jobs are assumed to have a certain laxity. Section 4 gives a slightly more complicated analysis of PARK, showing that the speed requirement of 5.828 can be reduced arbitrarily close to 1 when extra processors are used. Section 5 presents a simple lower bound result which shows that a nonmigratory schedule is not feasible on $m$ speed-$s$ processors if $s < 2m/(m+1)$. Also, the extension of PARK to firm-deadline scheduling is discussed.

**Notation.** Throughout this paper, we denote the release time, work (processing) time, and deadline of a job $J$ as $r(J)$, $p(J)$, and $d(J)$, respectively. We also assume that jobs have distinct deadlines (ties are broken using the job IDs). The laxity and span of $J$ are defined as $d(J) - r(J) - p(J)$ and $d(J) - r(J)$, respectively. For any $s \ge 1$, a speed-$s$ processor refers to a processor that can process $s$ units of work in one unit of time.

EDF refers to the strategy of scheduling jobs with earliest deadlines. Note that when a new job $J$ is released, the current job $J'$ on a processor will be preempted if $J'$ has a deadline later than $J$'s as well as the deadlines of the current jobs on all other processors. When we say an algorithm completes a job sequence, we mean that the algorithm completes all jobs in the sequence within their respective deadlines.

## 2. The PARK algorithm.

In this section, we describe a nonmigratory online algorithm called PARK and discuss several interesting properties of PARK. We will show in the next section that for any sequence $I$ of jobs that can be completed by a migratory offline schedule on $m$ unit-speed processors, $I$ can also be completed by PARK on $m$ speed-5.828 processors.

PARK is a very conservative algorithm. Roughly speaking, whenever PARK admits a job $J$ to a processor, it ensures that the processor has no commitment to other admitted jobs before the deadline of $J$. This concept of zero commitment is made formal through the following notion of *due*.

DEFINITION 1. *Let $t$ be the current time. Consider any job $J$.*

- *Denote $x_t(J)$ and $p_t(J)$ as the amount of work done on $J$ up to time $t$ and the remaining work, respectively (note that $p_t(J) = p(J) - x_t(J)$).*
- *Define the* latest processing interval (LPI) *of $J$ as the interval $[d(J) - p_t(J), d(J)]$. If $d(J) - p_t(J) < t$, we say that $J$* expires.
- *For any $t' > t$, define $due_t(J, t')$ as the minimum amount of $J$'s remaining work that a speed-1 processor needs to do by time $t'$ in order to complete $J$ by its deadline. More formally, $due_t(J, t') = \max\{0, t' - (d(J) - p_t(J))\}$ if $t' \le d(J)$, and $p_t(J)$ otherwise.*

PARK is a nonmigratory algorithm using $pm$ speed-$s$ processors, where $p$ and $m$ are positive integers and $s \ge 1$ is any real number. It maintains a central pool to store jobs that have been released but not yet admitted to any processor. PARK is nonmigratory and each processor has its own queue of admitted jobs. Before describing the algorithm, we have one more definition.

Definition 2. *Let $t$ be the current time. Consider any processor $P_i$ used by PARK. For any $t' > t$, we define $due_t(P_i, t')$ as the sum of $due_t(J, t')$ over each job $J$ in the queue of $P_i$ at time $t$.*

In the rest of the paper, we say that at time $t$, (i) a job $J$ has $w$ units of work due at time $t' > t$ if $due_t(J, t') = w$; (ii) a processor $P_i$ has $w$ units of work due at $t'$ if $due_t(P_i, t') = w$; (iii) a processor $P_i$ is doing some work due at $t'$ if $P_i$ is processing a job $J$ with $due_t(J, t') > 0$.

Algorithm 1 (PARK).

```
Job release: A job upon release is put into the pool if it cannot be admitted
to one of the processors.
Job admission: At any time $t$, let $S$ be the set of jobs in the pool plus possibly
the job just released at $t$. If $S$ is nonempty, we attempt to admit such jobs as
follows: Let $J \in S$ be the job with the earliest deadline.
• If $J$ expires, discard $J$.
• Else if there exists a processor $P_i$ such that $due_t(P_i, d(J)) = 0$, admit $J$ into
the queue of $P_i$.
Individual processor scheduling: Each processor schedules the jobs in its
queue using EDF.
```

It is worth mentioning that though PARK is using speed-$s$ processors, the definitions of $due_t(P_i, t')$ as well as LPI are based on a unit-speed processor. To understand the admission policy of PARK, we need to focus on the LPIs of the jobs. In general, the LPIs of jobs may overlap with each other. For example, if two jobs have the same deadline, their LPIs always share a common ending time. Yet the way PARK admits jobs aims to guarantee that, at any time, all jobs admitted to the same processor have nonoverlapping LPIs. This can be observed as follows. When a job $J$ is admitted by a processor $P_i$ at time $t$, all the previously admitted jobs have zero work due at $d(J)$. It means that at time $t$, the LPI of any previously admitted job starts no earlier than $d(J)$ and cannot overlap the LPI of $J$. As time passes, the LPI of each individual job might shrink (if $P_i$ has worked on it) but cannot get bigger; thus, the nonoverlapping property remains. The following is a precise statement about the nonoverlapping property. Figure 1 shows the LPIs of the jobs at different times when PARK is given a sequence of five jobs.

Property 1 (nonoverlapping property). *Let $t$ be the current time. Consider any processor $P_i$. Let $J_u$ and $J_v$ be two jobs in the queue of $P_i$. The LPIs of $J_u$ and $J_v$ do not overlap.*

The nonoverlapping property allows us to bound the commitment of each processor easily.

Property 2 (bounded-commitment property). *Let $t$ be the current time. Consider any processor $P_i$.*

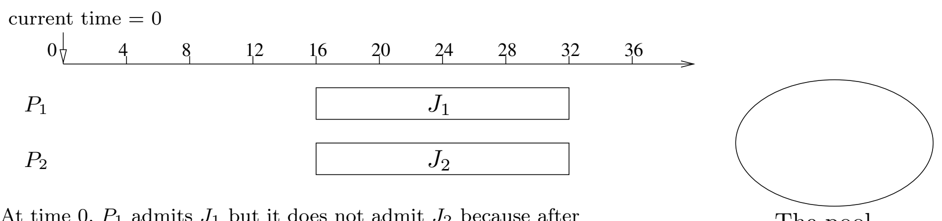
(a) *For every job in the queue of $P_i$, its LPI starts no earlier than $t$. That is, no job in the queue of $P_i$ expires.*

(b) *For any $t' > t$, $P_i$ has at most $(t' - t)$ units of work due at time $t'$.*

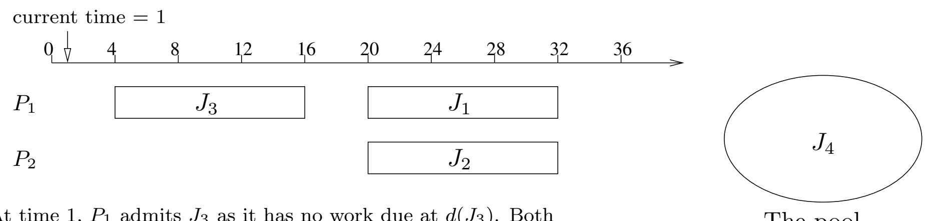
*Proof.* (a) For the sake of contradiction, assume that $J$ is the first job that expires in the queue of a processor $P_i$. When $J$ is admitted to $P_i$, $J$ has not yet expired. Let $t_0$ be the moment just before the first time $J$ is found to expire. That is, at $t_0$, the LPI of $J$ starts exactly at $t_0$; all other jobs in the queue of $P_i$ have not yet expired and their LPIs, due to the nonoverlapping property, must start after $J$'s LPI. In other

|  | Release time | Work | Deadline |
|---|---|---|---|
| $J_1$ | 0 | 16 | 32 |
| $J_2$ | 0 | 16 | 32 |
| $J_3$ | 1 | 12 | 16 |
| $J_4$ | 1 | 12 | 36 |
| $J_5$ | 2 | 12 | 28 |

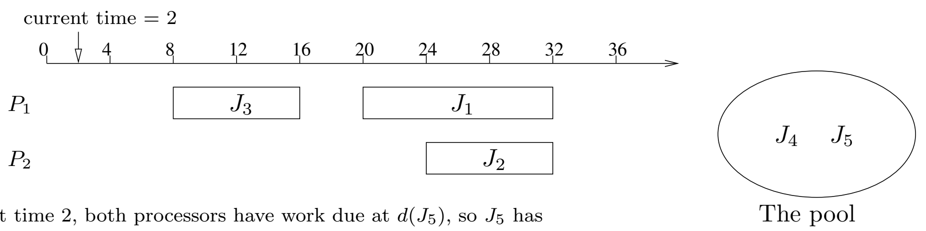
(a) Input job sequence



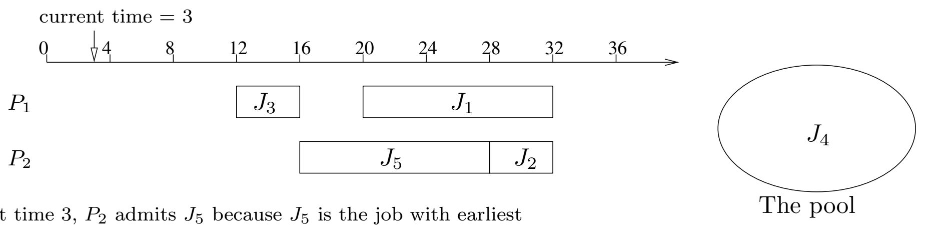
(b) At time 0, $P_1$ admits $J_1$ but it does not admit $J_2$ because after admitting $J_1$, $P_1$ has nonzero work due at $d(J_2)$. $P_2$ admits $J_2$.



(c) At time 1, $P_1$ admits $J_3$ as it has no work due at $d(J_3)$. Both processors have work due at $d(J_4)$, so $J_4$ has to wait in the pool.



(d) At time 2, both processors have work due at $d(J_5)$, so $J_5$ has to wait in the pool.



(e) At time 3, $P_2$ admits $J_5$ because $J_5$ is the job with earliest deadline in the pool, and $P_2$ has no work due at $d(J_5)$.

FIG. 1. *Figure* (a) *shows a sequence of five jobs to be scheduled by PARK on two speed-*4 *processors. Figures* (b)–(e) *show the LPIs (not the schedules) of the jobs in each processor at different times.*

words, all jobs except $J$ have deadlines later than $d(J)$. Recall that $P_i$ schedules jobs using EDF and its speed is $s \geq 1$. Thus, $J$ must be processed by $P_i$ at $t_0$ and cannot expire immediately after $t_0$. A contradiction occurs.

(b) At any time $t$, consider the LPIs of all the jobs in the queue of $P_i$. The LPIs are nonoverlapping and, by (a), the first LPI starts no earlier than $t$. For any $t' > t$, the amount of work due at $t'$ for $P_i$ is equal to the total length of each LPI (or its portion) that ends on or before $t'$. The latter is obviously upper bounded by $t' - t$. ∎

Since a job admitted by a processor $P_i$ never expires, it can be completed by $P_i$ by its deadline. In the next section, we will show that for a sequence of jobs that can be completed by some offline schedule, PARK will admit every job to a processor before they expire. PARK seems to be conservative in admitting jobs and often lets jobs wait in the pool. The next property shows that whenever a job is waiting, all processors are actually productive. In other words, such waiting is reasonable.

Property 3 (waiting property). *At any time $t$, if there is a job $J$ left in the pool, then*

- *all processors are busy, and*
- *all processors are doing some work due at $d(J)$.*

*Proof.* Suppose on the contrary that, at time $t$, there exists a processor $P_i$ that is idle. Then $P_i$ should admit $J$ or another job from the pool and become busy at $t$. A contradiction occurs. If $P_i$ at $t$ is working on a job $J_e$ that has no work due at $d(J)$, then the LPI of $J_e$ starts no earlier than $d(J)$. $P_i$ is using EDF and $J_e$ has the earliest deadline among all jobs in $P_i$. By the nonoverlapping property, at $t$, every job in $P_i$ has no work due at $d(J)$, and $P_i$ should admit $J$, or another job with a deadline earlier than $d(J)$, from the pool. This contradicts that at $t$, $P_i$ is working on $J_e$. ∎

## 3. Analysis of PARK.

In this section, we prove the main result that any job sequence that can be completed by some migratory offline schedule on $m$ speed-1 processors can also be completed by PARK on $m$ speed-$(3 + 2\sqrt{2})$ processors. Note that $3+2\sqrt{2} \approx 5.828$. We also show how to improve the resource bounds when jobs are assumed to have a certain laxity. For ease of discussion, this section will first present a lemma for the special case where every job has nonzero laxity, or equivalently, a work-span ratio strictly less than one. This is to illustrate the core idea of our analysis of PARK. Then we make use of a scaling technique to prove the general theorem, followed by two corollaries that capture the results stated earlier.

Lemma 3. *Let $I$ be any job sequence that can be completed by some migratory offline schedule on $m$ speed-1 processors. Let $0 < w < 1$. If all jobs in $I$ have a work-span ratio of at most $w$, then PARK can complete $I$ on $m$ speed-$(\frac{2}{1-w})$ processors.*

*Proof.* Note that the speed of the processors used by PARK is $\frac{2}{1-w} > 1$. As mentioned in the previous section, every job admitted by PARK to a processor can be completed by its deadline. To prove Lemma 3, it suffices to show that every job in $I$ is admitted rather than discarded in the admission step of PARK.

For the sake of contradiction, we assume that in the course of scheduling $I$ with PARK, some job expires in the pool and is discarded. Let $J$ be the first such job. Let $t_0 = d(J) - w(d(J) - r(J)) = r(J) + (1 - w)(d(J) - r(J))$. Since $J$ has a work-span ratio of at most $w$, $J$ cannot expire on or before $t_0$ and is in the pool during the time interval $[r(J), t_0]$.

Let $r(J) - l$ be the earliest time such that, at any time throughout the interval $[r(J) - l, t_0]$, there is at least one job in the pool with deadline on or before $d(J)$. Note that $l \geq 0$ is a real number. By the waiting property, during $[r(J) - l, t_0]$, all the $m$ processors of PARK are busy and the total work done by PARK is exactly

$$m \times \frac{2}{1-w} \times (t_0 - (r(J) - l)) = 2m(d(J) - r(J)) + \frac{2lm}{1-w}.$$

By the waiting property again, at any time in $[r(J) - l, t_0]$, PARK schedules all processors to do some work due at $d(J)$. Such work can be classified into two types:

1. Work owing to jobs that are admitted to processors before $r(J) - l$.
2. Work owing to jobs admitted during $[r(J) - l, t_0]$.

By the bounded-commitment property, at time $r(J) - l$, each processor has at most $d(J)-(r(J)-l)$ units of work due at $d(J)$, and hence the amount of type 1 work is at most $m \times (d(J) - (r(J) - l))$. Consider any job $J'$ admitted during $[r(J) - l, t_0]$. $J'$ has a deadline no later than $d(J)$ and, by definition of $l$, $J'$ must be released no earlier than $r(J) - l$. Note that the total work of jobs with release time at least $r(J)-l$ and deadline at most $d(J)$ cannot exceed $m(d(J)-(r(J)-l))$; otherwise such jobs cannot be completed by any offline schedule on $m$ unit-speed processors. On the other hand, $J$ is one of such jobs but is not admitted by PARK. The amount of type 2 work is at most $m \times (d(J) - (r(J) - l)) - p(J)$. In summary, the total amount of work done by PARK during $[r(J) - l, t_0]$ is at most

$$\begin{aligned}
&m(d(J) - (r(J) - l)) + m(d(J) - (r(J) - l)) - p(J) \\
&\quad < 2m(d(J) - r(J)) + 2lm \\
&\quad \leq 2m(d(J) - r(J)) + \frac{2lm}{1-w}.
\end{aligned}$$

This leads to a contradiction, and Lemma 3 follows. □

In the following, we present an extension to PARK so as to remove the assumption that the work-span ratio must be less than one. This extension also allows for a tradeoff between the processor speed and the number of processors used by PARK.

PARK($u$) is a scaled version of PARK characterized by a real number $u > 0$. Intuitively, PARK($u$) scales every job by a factor of $u$ and follows the schedules of PARK for the scaled jobs. When $u = 1$, PARK($u$) is identical to PARK. More specifically, to schedule a job sequence $I$ on $n$ speed-$s'$ processors, PARK($u$) simulates a copy of PARK that uses $n$ speed-$s$ processors, where $s = us'$. Whenever PARK($u$) receives a new job $J$, it releases a job $J'$ for PARK with $r(J') = r(J)$, $d(J') = d(J)$, and $p(J') = u \times p(J)$. Denote the processors used by PARK($u$) as $P_1, \ldots, P_n$ and those used by PARK as $P'_1, \ldots, P'_n$. At any time, PARK($u$) admits a job $J$ to a processor $P_i$ if PARK admits the corresponding job $J'$ to $P'_i$; PARK($u$) discards $J$ if PARK discards $J'$; and PARK($u$) runs a job $J$ on a processor $P_i$ if PARK runs $J'$ on $P'_i$.

We notice that PARK($u$) can always synchronize with the simulated copy of PARK because the amount of time for PARK($u$) to complete a job $J$ is exactly the same as that for PARK to complete the corresponding job $J'$. The following is the main theorem of this section.

THEOREM 4. *Let $I$ be any job sequence that can be completed by some migratory offline schedule on $m$ speed-1 processors. Let $0 < w \leq 1$. If all jobs in $I$ have a work-span ratio of at most $w$, then PARK($u$) can complete $I$ using $pm$ speed-$(\frac{p+u}{pu(1-wu)})$ processors, where $p$ is any positive integer and $u$ is any real number such that $0 < wu < 1$.*

Before proving Theorem 4, we illustrate how to choose the parameters in Theorem 4 so as to obtain the results claimed in the introduction. Consider the case where

$w = p = 1$. Choosing $u = \sqrt{2} - 1$ minimizes the speed requirement and gives the following corollary.

COROLLARY 5. *Let $I$ be any job sequence that can be completed by some migratory offline schedule on $m$ speed-1 processors. PARK$(u)$ with $u = \sqrt{2} - 1$ can complete $I$ using $m$ speed-$(3 + 2\sqrt{2})$ processors.*

Putting $u = 1/(2w)$ gives a more general corollary.

COROLLARY 6. *Let $I$ be any job sequence that can be completed by some migratory offline schedule on $m$ speed-1 processors. Let $0 < w \leq 1$. If all jobs in $I$ have a work-span ratio of at most $w$, then PARK$(u)$ with $u = 1/(2w)$ can complete $I$ using* (i) *$pm$ speed-$(4w + (2/p))$ processors for any positive integer $p$, or* (ii) *$\lceil 2/(s - 4w) \rceil m$ speed-$s$ processors for any $s > 4w$.*

*Proof of Theorem* 4. Recall that PARK$(u)$ uses $pm$ speed-$(\frac{p+u}{pu(1-wu)})$ processors. To schedule a job sequence $I$ as stated in the theorem, PARK$(u)$ simulates a copy of PARK that uses $pm$ speed-$(\frac{p+u}{p(1-wu)})$ processors. Let $I'$ be the sequence of the jobs created for PARK while PARK$(u)$ schedules $I$. As $I$ can be completed by some offline schedule on $m$ speed-1 processors, $I'$ can also be completed by some offline schedule on $m$ speed-$u$ processors. Each job in $I'$ has a work-span ratio of at most $wu$. To show that PARK$(u)$ can complete $I$, it suffices to show that PARK meets the deadlines of all jobs in $I'$. The proof is a straightforward generalization of Lemma 3.

First, note that $\frac{p+u}{p(1-wu)} > 1$, as both $u > 0$ and $1 > wu > 0$. Suppose, for the sake of contradiction, that PARK fails to complete a job in $I'$. This job must expire in the pool. Let $J$ be the first such job. Let $t_0 = r(J) + (1 - wu)(d(J) - r(J))$. As the work-span ratio of $J$ is at most $wu$, $J$ expires no earlier than $t_0$ and must have resided in the pool during the interval $[r(J), t_0]$. Again, let $l \geq 0$ be the largest number such that, at any time throughout the interval $[r(J) - l, t_0]$, there is at least one job in PARK's pool with deadline on or before $d(J)$. Note that $l \geq 0$. During $[r(J) - l, t_0]$, all $m$ processors are busy and the total work done by PARK is exactly

$$pm \times s \times (t_0 - (r(J) - l)) = (p + u)m(d(J) - r(J)) + \frac{(p + u)lm}{1 - wu}.$$

Furthermore, at any time in $[r(J) - l, t_0]$, every processor of PARK is always doing work due at $d(J)$ and such work belongs to either a job admitted before $r(J) - l$ or during $[r(J) - l, t_0]$. Thus, the total work done by PARK during $[r(J) - l, t_0]$ is at most

$$\begin{aligned} &pm(d(J) - (r(J) - l)) + mu(d(J) - (r(J) - l)) - p(J) \\ &\quad < (p + u)m(d(J) - r(J)) + l(p + u)m \\ &\quad \leq (p + u)m(d(J) - r(J)) + \frac{(p + u)lm}{1 - wu}. \end{aligned}$$

This leads to a contradiction, and PARK must be able to complete $I'$. Hence, the theorem follows. ☐

**4. Alternative analysis of PARK.** In this section, we show a more complicated analysis of PARK, resulting in the following theorem, which enables us to construct a nonmigratory online algorithm for hard-deadline scheduling that can exploit extra processors to reduce the speed requirement arbitrarily close to one (see Corollary 8).

THEOREM 7. *Let $0 < w < 1$ be a real number. Let $I$ be a job sequence that can be completed by some migratory offline schedule on $m$ speed-1 processors. If all jobs* *in $I$ have a work-span ratio of at most $w$, then $I$ can be completed by PARK using $pm$ speed-$s$ processors, where $p$ is any positive integer such that $p(1-w) > 1$ and $s = 1 + \frac{1}{p(1-w)-1}$.*

Using the scaling technique, we can remove the assumption of work-span ratios and extend Theorem 7 to any job sequence $I$. Recall that PARK($u$) scales every job of $I$ by a factor of $u$ and schedules the scaled jobs using PARK. Let $u = \frac{1}{1+\epsilon}$ for some $\epsilon > 0$, and denote the scaled version of $I$ as $I'$. Every job in $I'$ has a work-span ratio of at most $\frac{1}{1+\epsilon}$. By Theorem 7, $I'$ can be completed by PARK using $pm$ speed-$s$ processors where $s = 1 + \frac{1}{p(1-\frac{1}{1+\epsilon})-1}$. PARK($u$), using $pm$ speed-$s/u$ processors, can synchronize with PARK and complete $I$. In particular, choosing $p = \lceil (1+\frac{1}{\epsilon})^2 \rceil$, we have $s/u = (1 + \frac{1}{p(1-\frac{1}{1+\epsilon})-1})/u \le (1+\epsilon)^2$. This relationship is stated in the following corollary. Note that choosing a small $\epsilon$ means using more and slower processors.

COROLLARY 8. *Let $I$ be a job sequence that can be completed by some migratory offline schedule on $m$ speed-1 processors. Let $\epsilon > 0$ be any real number. PARK($u$) with $u = \frac{1}{1+\epsilon}$ can complete $I$ using $\lceil (1+\frac{1}{\epsilon})^2 \rceil \times m$ speed-$(1+\epsilon)^2$ processors.*

We are ready to prove Theorem 7. In the rest of this section, we assume that $I$ is a job sequence that can be completed by some migratory offline schedule OPT on $m$ speed-1 processors and all jobs in $I$ have a work-span ratio of at most $w < 1$. PARK is using $pm$ speed-$s$ processors, where $p$ is a positive integer such that $p(1-w) > 1$ and $s = 1 + \frac{1}{p(1-w)-1}$.

To prove Theorem 7, we need to compare the total amount of work due at any particular time in PARK and in OPT. At any time $t$, for any $t' > t$, we let $C(t, t')$ be the total amount of work due at $t'$ in PARK, and similarly $O(t, t')$ for OPT. Let $L(t, t') = C(t, t') - O(t, t')$. If $L(t, t') = \beta$, we say that at time $t$, PARK *lags behind* OPT by $\beta$ units of work due at $t'$. Furthermore, PARK is said to be *safe* at time $t$ if for any $t' > t$, $L(t, t')$ is proportional to the duration from $t$ to $t'$ (see the following definition).

DEFINITION 9. *In the course of scheduling $I$, at any time $t$, for any $t' > t$, PARK is $t'$*-safe *if $L(t, t') < \frac{m}{s-1}(t' - t)$. At any time $t$, PARK is* safe *if PARK is $t'$-safe for all $t' > t$.*

The most nontrivial observation in analyzing PARK is that at any time, PARK is safe (see Lemma 10). It is then relatively easy to see that, whenever a job $J$ is released to PARK, if PARK is safe at $r(J)$, then $J$ will be eventually admitted by PARK (see Lemma 11).

LEMMA 10. *In the course of scheduling $I$, at any time $t$, PARK is safe.*

LEMMA 11. *Let $J$ be any job in $I$. If PARK is safe at $r(J)$, then $J$ will be admitted by PARK on or before $t_0 = r(J) + (1-w)(d(J) - r(J))$.*

Lemmas 10 and 11 together guarantee that every job in $I$ must be admitted by PARK. As mentioned in section 2, PARK meets the deadlines of all admitted jobs. Thus, Theorem 7 follows. For ease of discussion, we will first present the proof for Lemma 11. We start with a more technical lemma, which will also be used in the proof of Lemma 10.

PROPOSITION 12. *Consider any time interval $[a, b]$ and any time $d > b$. Assume that PARK is $d$-safe at time $a$ and, at any time in $[a, b]$, there exists a job in PARK's pool with deadline no later than $d$. Then $b - a < \frac{1}{p(s-1)}(d - a)$. (Recall that $s$ is chosen as $1 + \frac{1}{p(1-w)-1}$; thus, $\frac{1}{p(s-1)} = 1 - w - \frac{1}{p}$.)*

*Proof.* Due to the waiting property, at any time in the interval $[a, b]$, all the $pm$ processors of PARK are busy and doing work due at $d$. The total work done by

PARK during $[a, b]$ is exactly

$$pm \times s \times (b - a). \tag{1}$$

PARK is $d$-safe at $a$ and $L(a, d) < \frac{m}{s-1}(d-a)$. At $a$, PARK and OPT have $C(a, d)$ and $O(a, d)$ units of work due at $d$, respectively. Consider all the jobs that are released from $a$ to $b$; the sum of their work due at $d$ is at most $m(d - a) - O(a, d)$ (otherwise, even OPT cannot complete these works by $d$). During the interval $[a, b]$, the total amount of work due at $d$ that PARK can possibly work on is at most

$$\begin{align} C(a, d) + m(d - a) - O(a, d) &= L(a, d) + m(d - a) \tag{2}\\ &< \frac{m}{s-1}(d - a) + m(d - a) \tag{3}\\ &= \frac{ms}{s-1}(d - a). \tag{4}\end{align}$$

Combining (1)–(4), we have $pms(b - a) < \frac{ms}{s-1}(d - a)$, or equivalently, $(b - a) < \frac{1}{p(s-1)}(d - a)$. ∎

Lemma 11 is in fact a corollary of Proposition 12. Details are as follows.

*Proof of Lemma* 11. Suppose on the contrary that at time $t_0 = r(J) + (1 - w)(d(J) - r(J))$, $J$ has not been admitted. As $J$ is assumed to have a work-span ratio of at most $w$, $J$ has not yet expired at $t_0$ and is in the pool during the interval $[r(J), t_0]$. Note that $t_0 - r(J) = (1 - w)(d(J) - r(J))$ and, by Proposition 12 (with $a = r(J)$, $b = t_0$, and $d = d(J)$), we have $(t_0 - r(J)) < (1 - w - \frac{1}{p})(d(J) - r(J))$. This implies that $\frac{1}{p} < 0$, contradicting that $p \geq 1$. ∎

The rest of this section is devoted to the proof of Lemma 10, which is further broken into two lemmas. We first state a simple fact about how the value of $L(t, t')$ changes over time.

FACT 1. *Let $[t_1, t_2]$ be a time interval before a certain time $t'$. Assume that during the interval $[t_1, t_2]$, PARK has done at least $x$ units of work due at $t'$ and OPT has done at most $y$ units of work due at $t'$. Then, $L(t_2, t') \leq L(t_1, t') - x + y$.*

Intuitively, we prove Lemma 10 inductively over time. Assume that PARK is safe at time $a$. We first consider two basic types of time intervals $[a, b]$ and show that, in either case, PARK is safe at time $b$. Precisely, for any $t' > b$, we call $[a, b]$

- a $t'$-quiet period if at any time in $[a, b]$ there is no job in PARK's pool with work due at $t'$, and
- a $t'$-hectic period if at any time in $[a, b]$ there is at least one job in PARK's pool with work due at $t'$.

During a $t'$-quiet period, any job with work due at $t'$ in PARK is admitted to some processor. Since PARK is using speed-$s$ processors, we can argue that PARK will not lag behind OPT too much on work due at $t'$ during a $t'$-quiet period (see Lemma 13). A hectic period is more complicated. In Lemma 14 we first show that with the extra speed and number of processors given to PARK, a $t'$-hectic period cannot last for too long. Then we notice that within such a short period, the amount of PARK's work due at $t'$ that lags behind OPT's cannot change too drastically and PARK is still $t'$-safe at $b$. Below we prove the above observations regarding quiet and hectic periods (see Lemmas 13 and 14). Then it is easy to show that PARK is safe at any time (i.e., Lemma 10).

LEMMA 13. *Consider a time interval $[a, b]$ and a certain time $t' > b$. Assume that at any time in the interval $[a, b]$ there is no job in PARK's pool with work due at $t'$. If PARK is safe at $a$, then PARK is $t'$-safe at $b$.*

*Proof.* We prove inductively that, at any time in $[a, b]$, PARK is $t'$-safe. Let $r > a$ be the first time a job is released; if no jobs are released on or before $b$, let $r = b$. Below we first show that PARK is $t'$-safe at time $r$, i.e., $L(r, t') \leq \frac{m}{s-1}(t' - r)$. Note that, at time $r$, if a job $J$ is released, it contributes exactly $p(J)$ to both $C(r, t')$ and $O(r, t')$ and does not affect the value of $L(r, t')$. To derive an upper bound of $L(r, t')$, it suffices to consider the amount of work released before $r$. In particular, we can tighten the trivial upper bound of $L(r, t')$ from $C(r, t')$ to $\hat{C}(r, t')$, where $\hat{C}(r, t')$ denotes, at time $r$, the amount of work in PARK that has been released before $r$ and due at $t'$.

Denote by $\Phi_r$ the set of PARK's processors which, at $r$, have work released before $r$ and due at $t'$. Then $\hat{C}(r, t') \leq |\Phi_r|(t' - r)$. If $|\Phi_r| < \frac{m}{s-1}$, then $L(r, t') \leq \hat{C}(r, t') < \frac{m}{s-1}(t' - r)$. Bound $L(r, t')$ for the case when $|\Phi_r| \geq \frac{m}{s-1}$ is more complicated. During $[a, r]$, the pool contains no job due at $t'$ and no job is released until $r$. Thus, at any time $t$ where $a \leq t < r$, if a processor of PARK does not have any work due at $t'$, this processor cannot be in $\Phi_r$. In other words, throughout the interval $[a, r]$, every processor in $\Phi_r$ has work due at $t'$ and PARK has done at least $|\Phi_r|s(r - a)$ units of work due at $t'$. Note that OPT achieves at most $m(r - a)$. Thus,

$$\begin{aligned} L(r, t') &\leq L(a, t') - |\Phi_r|s(r - a) + m(r - a) \\ &< \frac{m}{s-1}(t' - a) - \frac{m}{s-1}s(r - a) + m(r - a) \quad \text{(at time } a\text{, PARK is } t'\text{-safe)} \\ &= \frac{m}{s-1}(t' - r). \end{aligned}$$

In summary, we have proven that, at time $r$, PARK is $t'$-safe. If $r < b$, we can repeat the above argument to prove that PARK is $t'$-safe at each subsequent release time and eventually at time $b$. □

Next, we consider the case of hectic periods.

LEMMA 14. *Consider a time interval $[a, b]$ and a certain time $t' > b$. Assume that, just before $a$, there is no job in PARK's pool with work due at $t'$ and that, at any time in $[a, b]$, there is at least one job in PARK's pool with work due at $t'$. If PARK is safe at $a$, then*

(i) *$(b - a) < \frac{1}{s}(t' - a)$, and*

(ii) *PARK is $t'$-safe at $b$.*

*Proof.* (i) Due to the condition of Lemma 14, we know that, at any time in $[a, b]$, there is a job $J$ in the pool with work due at $t'$ and $J$ must be released on or after $a$. Due to the work-span ratio assumption, any job released after $a$ and with work due at $t'$ must have a deadline on or before $a + \frac{1}{1-w}(t' - a) \geq t'$. Let $d = a + \frac{1}{1-w}(t' - a)$. At time $a$, PARK is safe and, in particular, $d$-safe. By Proposition 12, $(b - a) < (1 - w - \frac{1}{p})(d - a) = (1 - w - \frac{1}{p})(t' - a)/(1 - w)$. Note that $s = 1 + \frac{1}{p(1-w)-1}$ and $\frac{1}{s} = \frac{p(1-w)-1}{p(1-w)} = (1 - w - \frac{1}{p})/(1 - w)$. Thus, $(b - a) < \frac{1}{s}(t' - a)$, and (i) follows.

(ii) Consider the $pm$ processors used by PARK. Let $\Psi$ be the set of PARK's processors which, at any time in the interval $[a, b]$, are doing work due at $t'$. Let $|\Psi| = \psi$. If $\psi \geq \frac{m}{s-1}$, then,

$$\begin{aligned} L(b, t') &\leq L(a, t') - \psi s(b - a) + m(b - a) \\ &< \frac{m}{s-1}(t' - a) - \frac{m}{s-1}s(b - a) + m(b - a) \\ &= \frac{m}{s-1}(t' - b). \end{aligned}$$

Next, we consider $\psi < \frac{m}{s-1}$. Label the processors in $\Psi$ as $P_1, P_2, \ldots, P_\psi$. At $a$, each of these processors has at most $t'-a$ units of work due at $t'$. Label the processors not in $\Psi$ as $P_{\psi+1}, \ldots, P_{pm}$. At $a$, for each processor $P_i$ not in $\Psi$, let $w_i$ be the amount of work due at $t'$. Let $W = \sum_{i=\psi+1}^{pm} w_i$. Then $L(a,t') \le C(a,t') \le \psi(t'-a) + W$.

From $a$ to $b$, each of $P_1, P_2, \ldots, P_\psi$, has done exactly $s(b-a)$ units of work due at $t'$. For each $P_i$, where $i = \psi+1, \ldots, pm$, $P_i$ at some point in $[a,b]$ is doing some work due at a time later than $t'$; thus $P_i$ has done at least $w_i$ units of work due at $t'$. In summary, during $[a,b]$, PARK, by time $b$, must have done at least $\psi \times s(b-a) + W$ units of work due at $t'$; note that OPT has done at most $m(b-a)$ units of work due at $t'$. Hence, we have the following conclusion:

$$\begin{aligned}
L(b,t') &\le L(a,t') - \Big(\psi s(b-a) + W\Big) + m(b-a) \\
&\le \psi(t'-a) + W - \Big(\psi s(b-a) + W\Big) + m(b-a) \\
&= \psi\Big(t' - a - s(b-a)\Big) + m(b-a) \\
&< \frac{m}{s-1}\Big(t' - a - s(b-a)\Big) + m(b-a) \\
&\qquad\qquad\qquad \text{(by Lemma 14(i), } (t'-a-s(b-a)) > 0) \\
&= \frac{m}{s-1}(t'-b).
\end{aligned}$$

In summary, no matter what the value of $\psi$ is, $L(b,t') < \frac{m}{s-1}(t'-b)$. Thus, PARK is $t'$-safe at time $b$. ◻

With the observations on the quiet and hectic periods, proving that PARK is safe at any time (i.e., Lemma 10) is straightforward.

*Proof of Lemma* 10. We first notice that, at time 0, $L(0,t')$ is equal to 0 for any $t' > 0$. Thus, PARK is safe at time 0. Let $\gamma_0 = 0$ and let $\gamma_1 = \min\{\gamma > \gamma_0 \mid$ at time $\gamma$, PARK switches from a $t'$-quiet period to a $t'$-hectic period for some $t' > \gamma\}$. Consider any time $t \le \gamma_1$. For any $t' > t$, $[\gamma_0, t]$ is a $t'$-quiet period and, by Lemma 13, PARK is $t'$-safe. Thus, PARK is safe at any time $t \le \gamma_1$. We can repeat the above argument to show inductively that PARK is safe at any time. In general, let $\gamma_{i+1} = \min\{\gamma > \gamma_i \mid$ at time $\gamma$, PARK switches from a $t'$-quiet period to a $t'$-hectic period for some $t' > \gamma$, or vice versa$\}$. Consider any time $t \le \gamma_{i+1}$. For any $t' > t$, let $j \le i$ be the smallest integer such that $[\gamma_j, t]$ is entirely a $t'$-quiet period or a $t'$-hectic period. By Lemma 13 and 14, PARK is $t'$-safe.

It is worth mentioning that, at any $\gamma_i$, a job is either released or admitted by PARK. Thus, in the course of scheduling $I$, there are only a finite number of $\gamma_i$'s. The above argument will complete eventually to show that PARK is safe at any time. ◻

## 5. Remarks.

**Lower bound.** Consider the following job sequence: $m+1$ identical jobs are released at time 0, each with $m$ units of work and deadline $m+1$. The set of jobs can be completed by a migratory schedule on $m$ speed-1 processors. For a nonmigratory (online or offline) schedule to complete the jobs on $m$ processors, some processor must admit at least two jobs, and thus the speed requirement is at least $\frac{2m}{m+1} = 2 - \frac{2}{m+1}$.

**Firm-deadline schedule.** Recall that in the firm-deadline scheduling problem, there may be too many jobs to be completed, and failing to complete a job causes only a loss in value due to that job and does not cause a system failure. Given a set $I$ of such jobs, the objective of a scheduler is to maximize the value obtained from completing the jobs. An online algorithm is said to be $c$-competitive for some $c \geq 1$ if, for any job sequence $I$, the algorithm can obtain at least a fraction of $1/c$ of the value obtained by the optimal offline schedule on $m$ speed-1 processors.

Consider the special case when the value of a job is proportional to its processing time. If migration is allowed, EDF-AC (EDF with admission control) using $m$ speed-3 processors is 1-competitive [12]. Since EDF-AC decides whether to complete or discard a job once the job is released, we can use it to select jobs for scheduling in PARK without migration. The actual operation is as follows. Whenever EDF-AC decides to complete a job $J$, we release $J$ to PARK with $p(J)$ scaled down to $p(J)/3$. The job sequence selected by EDF-AC can be completed by $m$ speed-3 processors, so the scaled job sequence can be completed by $m$ speed-1 processors and all jobs in the scaled sequence have work-span ratio of at most 1/3. By Corollary 6 with $p = 1$ and $w = 1/3$, the scaled job sequence can be completed by PARK using $m$ speed-$(\frac{10}{3})$ processors. As any job in the scaled sequence has only one-third of the original work, to complete the job actually selected by EDF-AC, we further increase the speed of the processors by a factor of 3. In summary, EDF-AC plus PARK gives a new algorithm which, using $m$ speed-10 processors, is 1-competitive nonmigratory for the firm-deadline scheduling problem.

**Effect of laxity on $\boldsymbol{\omega_m}$.** Consider the offline scheduling problem. Recall that $\omega_m$ is the maximum ratio, over all possible inputs, between the value attained by the optimal migratory schedule and that attained by the optimal nonmigratory schedule. The analysis of PARK reveals some information about the value of $\omega_m$ when all jobs are assumed to have a certain amount of laxity. More precisely, if all jobs have a work-span ratio no greater than $w$, where $w < \frac{1}{4}$, Corollary 6 shows that, for any job sequence, the subset of jobs that can be completed by the optimal offline migratory schedule on $m$ processors can also be completed by $\lceil 2/(1-4w) \rceil m$ (unit-speed) processors. By selecting the $m$ processors that achieve the highest values, we obtain a nonmigratory offline schedule attaining a value of at least $\frac{1}{\lceil 2/(1-4w) \rceil}$ of the value of the optimal migratory schedule. Hence, $\omega_m \leq \lceil 2/(1-4w) \rceil$.

**Implementation of PARK.** We notice that PARK admits a simple distributed implementation which does not require a centralized scheduler. Instead, each processor can monitor the pool and admit a job according to its own status, i.e., each processor does not need to inquire about the status of other processors. This is different from many other scheduling algorithms (e.g., EDF, LLF) in which the status of all processors is needed in order to make a scheduling decision. Thus, PARK is particularly useful when it is difficult to obtain complete information about all processors.

**Open problems.** Let $I$ be a job sequence that can be completed by some migratory offline schedule on $m$ speed-1 processors. Consider the processor speed required to obtain a nonmigratory online schedule for $I$. There is a gap between the upper bound of 5.828 and the lower bound of $2 - \frac{2}{m+1}$. The current analysis of PARK seems to be quite loose and we believe that a better analysis could possibly reduce the speed requirement to 4. We have shown that when extra processors are given, the speed requirement of PARK can be reduced arbitrarily close to 1. However, we do not know of any (migratory or nonmigratory) online algorithm that can guarantee the completion of $I$ using only $f(m)$ speed-1 processors, where $f(m)$ is a function of $m$. For the problem of firm-deadline scheduling, the current analysis depends on EDF-AC as the job selection module. In fact, we conjecture that PARK alone (say, with $m$ speed-9 processors) is sufficient to match the performance of any offline schedule.

## REFERENCES

[1] S. Baruah, G. Koren, B. Mishra, A. Raghunathan, L. Rosier, and D. Shasha, *On-line scheduling in the presence of overload*, in Proceedings of the IEEE Real-Time Systems Symposium, 1991, pp. 101–110.

[2] P. Berman and C. Coulston, *Speed is more powerful than clairvoyance*, Nordic J. Comput., 6 (1999), pp. 181–193.

[3] M. Chrobak, L. Epstein, J. Noga, J. Sgall, R. van Stee, T. Tichý, and N. Vakhania, *Preemptive scheduling in overloaded systems*, J. Comput. System Sci., 67 (2003), pp. 183–197.

[4] M. L. Dertouzos, *Control robotics: The procedural control of physical processes*, in Proceedings of the IFIP Congress, North-Holland, Amsterdam, 1974, pp. 807–813.

[5] M. L. Dertouzos and A. K. L. Mok, *Multiprocessor on-line scheduling of hard-real-time tasks*, IEEE Trans. Software Engrg., 15 (1989), pp. 1497–1506.

[6] J. Edmonds, *Scheduling in the dark*, Theoret. Comput. Sci., 235 (2000), pp. 109–141.

[7] B. Kalyanasundaram and K. R. Pruhs, *Speed is as powerful as clairvoyance*, J. ACM, 47 (2000), pp. 617–643.

[8] B. Kalyanasundaram and K. R. Pruhs, *Eliminating migration in multi-processor scheduling*, J. Algorithms, 38 (2001), pp. 2–24.

[9] G. Koren, E. Dar, and A. Amir, *The power of migration in multiprocessor scheduling of real-time systems*, SIAM J. Comput., 30 (2000), pp. 511–527.

[10] G. Koren and D. Shasha, *MOCA: A multiprocessor on-line competitive algorithm for real-time system scheduling*, Theoret. Comput. Sci., 128 (1994), pp. 75–97.

[11] T. W. Lam and K. K. To, *Trade-offs between speed and processor in hard-deadline scheduling*, in Proceedings of the Tenth Annual ACM-SIAM Symposium on Discrete Algorithms, SIAM, Philadelphia, ACM, New York, 1999, pp. 623–632.

[12] T. W. Lam and K. K. To, *Performance guarantee for EDF under overload*, J. Algorithms, 52 (2004), pp. 193–206.

[13] C. A. Phillips, C. Stein, E. Torng, and J. Wein, *Optimal time-critical scheduling via resource augmentation*, Algorithmica, 32 (2002), pp. 163–200.

[14] K. Pruhs, J. Sgall, and E. Torng, *Online scheduling*, in Handbook of Scheduling: Algorithms, Models and Performance Analysis, J. Leung, ed., CRC Press, Boca Raton, FL, 2004, pp. 15-1–15-41.

[15] J. A. Stankovic, M. Spuri, K. Ramamritham, and G. C. Buttazzo, *Deadline Scheduling for Real-Time Systems: EDF and Related Algorithms*, Kluwer Academic Publishers, Dordrecht, 1998.

SIAM J. COMPUT. © 2005 Society for Industrial and Applied Mathematics
Vol. 34, No. 3, pp. 683–696

# ON EVEN TRIANGULATIONS OF 2-CONNECTED EMBEDDED GRAPHS*

HUAMING ZHANG† AND XIN HE†

**Abstract.** Recently, Hoffmann and Kriegel proved an important combinatorial theorem [*SIAM J. Discrete Math.*, 9 (1996), pp. 210–224]: Every 2-connected bipartite plane multigraph $G$ without 2-cycle faces has a triangulation in which all vertices have even degree (this is called an *even triangulation*). Combined with the classical Whitney's theorem, this result implies that every such graph has a 3-colorable plane triangulation. Using this theorem, Hoffmann and Kriegel significantly improved the upper bounds of several art gallery and prison guard problems. A complicated $O(n^2)$ time algorithm was obtained in [*SIAM J. Discrete Math.*, 9 (1996), pp. 210–224] for constructing an even triangulation of $G$. Hoffmann and Kriegel conjectured that there is an $O(n^{3/2})$ time algorithm for solving this problem.

In this paper, we develop a simple proof of the above theorem. Our proof reveals and relies on a natural correspondence between even triangulations of $G$ and certain orientations of $G$. Based on this new proof, we obtain a very simple $O(n)$ time algorithm for finding an even triangulation of $G$. We also extend our proof to show the existence of even triangulations for similar graphs on high genus surface.

**Key words.** plane graph, even triangulations, graph coloring, high genus graph

**AMS subject classifications.** 05C07, 05C15, 05C85, 05C90, 68R10

**DOI.** 10.1137/S0097539702408247

## 1. Introduction.

Let $G = (V, E)$ be a 2-connected bipartite plane multigraph without 2-cycle faces. A *triangulation* of $G$ is a plane graph obtained from $G$ by adding new edges into the faces of $G$ so that all of its faces are triangles. A triangulation $G'$ of $G$ is called *even* if all vertices of $G'$ have even degree. Recently, Hoffmann and Kriegel proved an important combinatorial theorem [3, 4].

THEOREM 1. *Every 2-connected bipartite plane multigraph without 2-cycle faces has an even triangulation.*

Combined with the following classical Whitney's theorem, Theorem 1 implies that every 2-connected bipartite plane multigraph has a 3-colorable plane triangulation.

THEOREM 2. *A plane triangulation is 3-colorable iff all of its vertices have even degree.*

An elegant proof of Theorem 2 can be found in [10].

Note: In the statement of Theorem 1 in [3, 4], the phrase "without 2-cycle faces" was not explicitly mentioned. It is implicitly used in their paper. However, without this condition, Theorem 1 is not true. To be precise, we explicitly mention this condition in Theorem 1.

In addition to its importance in graph theory, Hoffmann and Kriegel showed that, based on Theorem 1, the upper bounds of several *art gallery* and *prison guard* problems (a group of problems extensively studied in computational geometry; see [11, 12]) can be significantly improved [4]. Theorem 1 was proved in [4] by showing that a linear equation system derived from the input graph $G$ has a solution. An

*Received by the editors May 22, 2002; accepted for publication (in revised form) September 10, 2004; published electronically March 17, 2005. The research of the second author was supported in part by NSF grant CCR-0309953.

http://www.siam.org/journals/sicomp/34-3/40824.html

†Department of Computer Science and Engineering, State University of New York at Buffalo, Buffalo, NY 14260 (huazhang@cse.buffalo.edu, xinhe@cse.buffalo.edu).

even triangulation of $G$ is found by solving this linear equation system. By using the *generalized nested dissection technique* of Lipton, Rose, and Tarjan [9], Hoffmann and Kriegel obtained a complicated algorithm with $O(n^2)$ runtime. This algorithm is the bottleneck of the algorithms for solving the art gallery and prison guard application problems discussed in [4]. By showing that another linear equation system derived from $G$ has a solution, Hoffmann and Kriegel discovered a relationship between all even triangulations of $G$ and certain closed walks in the dual graph of $G$.

It was conjectured in [4] that an even triangulation of $G$ can be found in $O(n^{1.5})$ time. In this paper, we give a very different proof of Theorem 1 without relying on the linear equation system derived in [4]. Our new proof is based on the discovery of a natural one-to-one correspondence between the set of all even triangulations of $G$ and the set of certain orientations of $G$ and its dual graph $G^*$. This new proof leads to a *very simple* $O(n)$ time algorithm for constructing an even triangulation of $G$. Thus, all algorithms for solving the art gallery and prison guard problems discussed in [4] can be uniformly improved to linear time. As a by-product, we also obtain a simple formula for calculating the number of distinct even triangulations of $G$.

A natural question is whether these theorems hold true for similar graphs on high genus surfaces. We show that our new proof of Theorem 1 and the algorithm for constructing even triangulations of 2-connected bipartite plane multigraphs without 2-cycle faces also apply to similar graphs on high genus surfaces. However, because a crucial property that is true for plane graphs does not hold for graphs on high genus surfaces, we can only prove a lower bound on the number of distinct even triangulations of such graphs. The problem of determining the exact value of this number remains an open problem.

The rest of the paper is organized as follows. In section 2, we introduce the definitions and preliminary results in [3, 4]. In section 3, we provide new proofs of Theorem 1 and another key theorem in [3, 4]. In section 4, we present our algorithm for finding even triangulations of plane graphs. In section 5, we investigate the problem for graphs on high genus surfaces.

**2. Preliminaries.** In this section, we give definitions and preliminary results. All definitions are standard and can be found in [1]. Let $G = (V, E)$ be a graph with $n = |V|$ vertices and $m = |E|$ edges. A *multigraph* is a graph where two edges can share the same pair of end vertices. We will restrict ourselves to multigraph without self loops in this paper, which will be simply called graph from now on, unless otherwise specified. The *degree* of a vertex $v \in V$, denoted by $deg_G(v)$, is the number of edges incident to $v$. If $G$ is clearly understood, we simply write $deg(v)$ for $deg_G(v)$. For a subset $V_1 \subseteq V$, $G - V_1$ denotes the graph obtained from $G$ by deleting the vertices in $V_1$ and their incident edges. A vertex $v$ of a connected graph $G$ is called a *cut vertex* if $G - \{v\}$ is disconnected. $G$ is 2-*connected* if it has no cut vertices. $G = (V, E)$ is *bipartite* if its vertex set $V$ can be partitioned into two subsets $V_1$ and $V_2$ such that no two vertices in $V_1$ are adjacent and no two vertices in $V_2$ are adjacent. A $k$-*coloring* of $G$ is a coloring of $V$ by $k$ colors so that no two vertices with the same color are adjacent to each other. Note that $G$ is bipartite iff it's 2-colorable.

A *cycle* $C$ of $G$ is a sequence of distinct vertices $u_1, u_2, \ldots, u_k$ such that $(u_i, u_{i+1}) \in E$ for $1 \leq i < k$ and $(u_k, u_1) \in E$. We also use $C$ to denote the set of the edges in it. If $C$ contains $k$ edges, it is called a $k$-cycle. A 3-cycle is also called a *triangle*. A *closed walk* is similarly defined except that it allows repeated vertices (but not repeated edges).

A *plane* graph $G$ is a graph embedded in the plane without edge crossings (i.e., an embedded planar graph). The embedding of a plane graph $G$ divides the plane into a number of regions. All regions are called *faces*. The unbounded region is called the *exterior face*. Other regions are called *interior faces*. The *degree* of a face is the number of edges on its boundary.

The dual graph $G^* = (V^*, E^*)$ of a plane graph $G$ is defined as follows: For each face $f$ of $G$, $V^*$ contains a vertex $v_f$. For each edge $e$ in $G$, $G^*$ has an edge $e^* = (v_{f_1}, v_{f_2})$, where $f_1$ and $f_2$ are the two faces of $G$ with $e$ on their common boundary. $e^*$ is called the *dual edge* of $e$. The mapping $e \Leftrightarrow e^*$ is a one-to-one correspondence between $E$ and $E^*$.

A *diagonal* of a plane graph $G$ is an edge which does not belong to $E$ and connects two vertices of a facial cycle. $G$ is *triangulated* if it has no diagonals. (Namely, all of its facial cycles, including the exterior face, are triangles.) A *triangulation* of $G$ is obtained from $G$ by adding a set of diagonals such that the resulting graph is plane and triangulated. An *even triangulation* is a triangulation in which all vertices have even degree. In this case, we also call the set of diagonals added into $G$ an even triangulation of $G$. The overloading of the term "even triangulation" makes our discussion easier. The meaning of "even triangulation" is always clear from context. Obviously, in order to triangulate a plane graph $G$, we cannot allow 2-cycle faces in $G$. Therefore, we will exclude the existence of 2-cycle faces in $G$ in this paper from now on. Thus, a plane graph really means a plane multigraph without 2-cycle faces.

For a 2-connected bipartite plane graph $G$, all of its facial cycles have even length. In order to triangulate $G$, we first add diagonals into the faces of $G$ so that all facial cycles of the resulting graph are 4-cycles. This can be easily done in linear time. Thus, without loss of generality, $G$ always denotes a 2-connected bipartite plane graph all of whose facial cycles have length 4. Such a graph will be called a 2-connected maximal bipartite plane graph (2MBP graph, for short). By Euler's formula, a 2MBP graph with $n$ vertices always has $n-2$ faces and $m = 2n-4$ edges. We denote the faces of $G$ by $Q(G) = \{q_1, q_2, \dots, q_{n-2}\}$. When $G$ is clearly understood, we simply use $Q$ to denote $Q(G)$.

Since $G$ is bipartite, we can fix a 2-coloring of $G$ with colors 0 and 1. For any face $q_i \in Q$, the set of the four vertices on the boundary of $q_i$ is denoted by $V_{q_i}$ and we set $Q_v = \{q_i \in Q | v \in V_{q_i}\}$. Since every facial cycle of $G$ is a 4-cycle, every face $q_i \in Q$ has two diagonals: the diagonal joining the two 0-colored (1-colored, respectively) vertices in $V_{q_i}$ is called the 0-diagonal (1-diagonal, respectively). Thus a triangulation of $G$ is nothing but choosing for each face $q_i$ either the 0-diagonal or the 1-diagonal and adding it into $q_i$. We associate each face $q_i \in Q$ with a $\{0,1\}$-valued variable $x_i$, and each triangulation $T$ of $G$ with a vector $\vec{x} = (x_1, x_2, \dots, x_{n-2}) \in GF(2)^{n-2}$, where

$$T \text{ contains the 0-diagonal of the face } q_i \Longleftrightarrow x_i = 1.$$

This mapping defines a one-to-one correspondence between the set of triangulations of $G$ and $GF(2)^{n-2}$. We choose $GF(2)$ because most calculations in this section are interpreted in $GF(2)$.

*Observation.* If $c(v)$ is the color of a vertex $v \in V_{q_i}$, then the term $x_i + c(v) \pmod 2$ describes the increase of the degree of $v$ after adding the diagonal of $q_i$ which corresponds to the value $x_i$ into the face $q_i$. Based on this observation, Hoffmann and Kriegel [3, 4] proved that a vector $\vec{x} = (x_i)_{1 \le i \le n-2} \in GF(2)^{n-2}$ represents an even triangulation of $G$ iff $\vec{x}$ is a solution of the following linear equation system (with $n$

equations and $n-2$ variables) over $GF(2)$:

$$\sum_{q_i \in Q_v} x_i = deg(v) + |Q_v|c(v) \pmod 2 \quad (\forall v \in V). \tag{1}$$

In [4], Hoffmann and Kriegel showed that (1) always has a solution, and hence proved Theorem 1. Therefore, the term "a vector $\vec{x}$ representing an even triangulation of $G$" and the term "a solution vector of (1)" can be used interchangeably.

To obtain all solutions of (1) (i.e., all even triangulations of $G$), [4] introduced the concept of *straight walk*. For a 2MBP graph $G$, its dual graph $G^*$ is 4-regular and connected. Consider a walk $S$ in $G^*$. Since $G^*$ is 4-regular, at every vertex of $S$, we have four possible choices to continue the walk: go back, turn left, go straight, or turn right. A closed walk of $G^*$ consisting of only straight steps at every vertex is called a *straight walk*, or *S-walk*. The edge set of $G^*$ can be uniquely partitioned into $S$-walks. We use $\mathcal{S}(G^*) = \{S_1, \ldots, S_k\}$ to denote this partition, where each $S_i$ $(1 \le i \le k)$ is an $S$-walk of $G^*$. Each vertex of $G^*$ (i.e., each face of $G$) occurs either twice on one $S$-walk or on two different $S$-walks. If a face $f$ occurs on one $S$-walk twice, it is called a 1-*walk face*. If $f$ occurs on two different $S$-walks, it is called a 2-*walk face*.

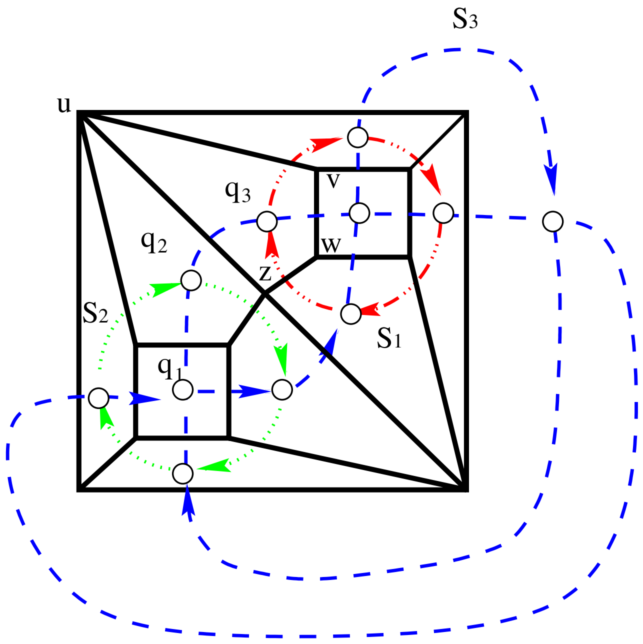


FIG. 1. *A 2MBP graph $G$, the dual graph $G^*$, and one of its $S$-orientation $\mathcal{O}$.*

Figure 1 shows a 2MBP graph and its dual graph $G^*$. The edges of $G$ are represented by solid lines. The edges of $G^*$ are represented by dotted lines. The vertices of $G^*$ (i.e., the faces of $G$) are represented by small circles. $\mathcal{S}(G^*)$ contains 3 $S$-walks $S_1$, $S_2$, and $S_3$. The face $q_1$ is a 1-walk face since it occurs on $S_3$ twice. The face $q_2$ is a 2-walk face since it occurs on both $S_2$ and $S_3$. (The $S$-walks in this figure are directed. Its meaning will be discussed in the next section.)

Let $S$ be an $S$-walk of $G^*$. Take an even triangulation of $G$. If we flip the diagonals of every face on $S$, it's easy to see that we get another even triangulation of $G$. (If a face $f$ occurs on $S$ twice, its diagonal is flipped twice and hence remains unchanged.)

THEOREM 3. *Let $G$ be a 2MBP graph and let $\mathcal{S}(G^*) = \{S_1, \ldots, S_k\}$ be the $S$-walks of $G^*$. The following statements hold:*

1. *If $T$ is an even triangulation of $G$ and the diagonals of $T$ are flipped along an $S$-walk $S_i$, we obtain another even triangulation of $G$.*
2. *If $T_1$ and $T_2$ are two even triangulations of $G$, there is a collection of $S$-walks such that by flipping the diagonals of $T_1$ along these $S$-walks we obtain $T_2$.*

Statement 1 of Theorem 3 was proved in [4]. Statement 2 of Theorem 3 was stated in [4]. Hoffmann and Kriegel mentioned that statement 2 can be proved by showing that another linear equation system derived from $G$ has a solution. However, since "the arguments are much more involved," its proof was omitted in [4] and was given in the technical report [3].

**3. Even triangulations of 2MBP graphs.** In this section, we provide new proofs of Theorems 1 and 3. First, we introduce a few definitions. An *orientation* of an undirected graph is an assignment of directions to its edges.

DEFINITION 1. *Let $G$ be a 2MBP graph.*

1. *A* $G$-orientation *of $G$ is an orientation of $G$ such that every facial cycle of $G$ is decomposed into two directed paths, one in a clockwise direction and another in a counterclockwise direction, each of length 2.*
2. *Let $\mathcal{G}_1, \mathcal{G}_2$ be two $G$-orientations of $G$. If every edge of $G$ has reverse direction in $\mathcal{G}_1$ and $\mathcal{G}_2$, we say $\mathcal{G}_2$ is the* reverse *of $\mathcal{G}_1$. In this case, $\mathcal{G}_1$ and $\mathcal{G}_2$ are called a* $G$-orientation pair *of $G$.*
3. *Fix a $G$-orientation $\mathcal{G}$ of $G$. For each face $q$ of $G$, the starting and the ending vertices of the two directed paths on the boundary of $q$ are called the* primary vertices *of $q$. The diagonal of $q$ connecting the two primary vertices is called its* primary diagonal *(with respect to $\mathcal{G}$). The other diagonal of $q$ is called its* secondary diagonal *(with respect to $\mathcal{G}$).*

It is worth mentioning that $G$-orientation should not be confused with the well-known *st-orientation*, which is widely used in graph theory and algorithms. (For its definition and applications, see, for example, [8, 13].) First of all, any *st*-orientation of a 2-connected plane graph has one and only one global source and sink, while this is not necessarily true in the case of $G$-orientation (although in $G$-orientation, each face has one source and one sink, which is similar to *st*-orientation). Second, in *st*-orientation, every facial cycle is also decomposed into two directed paths, but their source and sink could be adjacent, which is forbidden in $G$-orientation. We will see that this requirement plays a crucial role in our construction of even triangulations later on.

Note that for a $G$-orientation pair $\mathcal{G}_1, \mathcal{G}_2$ and any face $q$ of $G$, the primary diagonal of $q$ with respect to $\mathcal{G}_1$ is the same as its primary diagonal with respect to $\mathcal{G}_2$. If $\mathcal{G}_1, \mathcal{G}_2$ do not form a $G$-orientation pair, then for some faces $q$ of $G$, the primary diagonal of $q$ with respect to $\mathcal{G}_1$ is different from that with respect to $\mathcal{G}_2$.

DEFINITION 2. *Let $G$ be a 2MBP graph and let $G^*$ be its dual graph with $\mathcal{S}(G^*) = \{S_1, \ldots, S_k\}$.*

1. *An* $S$-orientation *of $G^*$ is an orientation of $G^*$ such that every $S$-walk $S_i$ $(1 \leq i \leq k)$ is a directed closed walk.*
2. *Let $\mathcal{O}_1, \mathcal{O}_2$ be two $S$-orientations of $G^*$. If every $S_i$ $(1 \leq i \leq k)$ has reverse direction in $\mathcal{O}_1$ and $\mathcal{O}_2$, we say $\mathcal{O}_2$ is the* reverse *of $\mathcal{O}_1$. In this case, $\mathcal{O}_1$ and $\mathcal{O}_2$ are called an* $S$-orientation pair *of $G^*$.*
3. *Fix an $S$-orientation $\mathcal{O}$ of $G^*$. For each face $q$ of $G$, if an $S$-walk $S_i$ of $G^*$*

*steps out of (into, respectively) $q$ through an edge $e$ on the boundary of $q$, then $e$ is called an* out-edge *(*in-edge*, respectively) of $q$ (with respect to $\mathcal{O}$).*

We can assign two different directions to each $S$-walk. So, if $G^*$ has $k$ $S$-walks, it has $2^k$ distinct $S$-orientations and $2^{k-1}$ distinct $S$-orientation pairs.

Consider an arbitrary $S$-orientation $\mathcal{O}$ of $G^*$. It is easy to check that every face $q$ of $G$ has two out-edges and two in-edges with respect to $\mathcal{O}$, and the two in-edges of $q$ are always incident to a common vertex on the boundary of $q$. Thus there are always two nonadjacent vertices on the boundary of $q$ that are incident to both in-edges and out-edges of $q$. For example, consider the face $q_3$ in Figure 1. An $S$-orientation $\mathcal{O}$ of $G^*$ is shown in the figure. With respect to $\mathcal{O}$, the edge $(u, v)$ is an out-edge of $q_3$ and the edge $(w, z)$ is an in-edge of $q_3$. The vertices $u$ and $w$ are incident to both in-edges and out-edges of $q_3$.

Next we show that there exists a natural one-to-one mapping between the set of $S$-orientations of $G^*$ and the set of $G$-orientations of $G$ which preserves the pair relation.

Definition 3. *Let $G$ be a 2MBP graph and let $\mathcal{O}$ be an $S$-orientation of $G^*$. We define an orientation of $G$ from $\mathcal{O}$ as follows. Let $e$ be any edge of $G$ and let $e^*$ be its dual edge in $G^*$. Let $q_1$ and $q_2$ be the two faces of $G$ with $e$ on their boundaries. Suppose that $e^*$ is directed from $q_2$ to $q_1$ in $\mathcal{O}$. When traveling $e^*$ from $q_2$ to $q_1$, we direct $e$ from right to left. (In other words, $e$ is directed counterclockwise on the boundary of $q_2$ and clockwise on the boundary of $q_1$.) This orientation of $G$ will be called the* orientation induced *from $\mathcal{O}$ and denoted by $\pi(\mathcal{O})$.*

An example of the induced orientation is shown in Figure 2(a). The $S$-walks $S_i, S_j, S_m$ pass through the faces $q_1, q_2$ in the shown directions. The induced directions of the edges on the boundaries of $q_1$ and $q_2$ are also shown.

Lemma 1. *Let $G$ be a 2MBP graph. If $\mathcal{O}$ is an $S$-orientation of $G^*$, then $\pi(\mathcal{O})$ is a $G$-orientation of $G$. If $\mathcal{O}$ and $\mathcal{O}'$ form an $S$-orientation pair, then $\pi(\mathcal{O})$ and $\pi(\mathcal{O}')$ form a $G$-orientation pair of $G$.*

*Proof.* Let $\mathcal{O}$ be an $S$-orientation of $G^*$. For any face $q$ of $G$, its two in-edges with respect to $\mathcal{O}$ are incident and hence form a directed path of length 2, in a clockwise direction, in $\pi(\mathcal{O})$. Its two out-edges with respect to $\mathcal{O}$ are also incident and hence form a directed path of length 2, in a counterclockwise direction, in $\pi(\mathcal{O})$. Thus $\pi(\mathcal{O})$ is a $G$-orientation of $G$.

Let $\mathcal{O}$ and $\mathcal{O}'$ be an $S$-orientation pair of $G^*$. Let $\mathcal{G} = \pi(\mathcal{O})$ and $\mathcal{G}' = \pi(\mathcal{O}')$. The direction of each $S$-walk in $\mathcal{O}'$ is the reverse of that in $\mathcal{O}$. So for any face $q$ of $G$, its two *in-edges* with respect to $\mathcal{O}$ are the two *out-edges* with respect to $\mathcal{O}'$. Therefore, the direction of each edge $e$ in $\mathcal{G}'$ is the reverse of that in $\mathcal{G}$. Hence $\mathcal{G}$ and $\mathcal{G}'$ form a $G$-orientation pair. ∎

Let $\mathcal{O}$ be an $S$-orientation of $G^*$ and $\mathcal{G} = \pi(\mathcal{O})$. For any face $q$ of $G$, the primary diagonal $d$ of $q$ with respect to $\mathcal{G}$ is the diagonal connecting the two vertices on the boundary of $q$ that are incident to both the in-edges and the out-edges of $q$ with respect to $\mathcal{O}$. We also call $d$ the primary diagonal of $q$ with respect to $\mathcal{O}$. For example, consider the face $q_1$ in Figure 2(a). The vertices $v_1$ and $v_3$ are incident to both in-edges and out-edges of $q_1$. Thus the primary diagonal of $q_1$ with respect to $\mathcal{O}$ is $(v_1, v_3)$.

Note that if $q$ is a 1-walk face that occurs on an $S$-walk $S_i$ twice and if the direction of $S_i$ is reversed, the primary diagonal of $q$ remains unchanged. On the other hand, consider a 2-walk face $q$ that occurs on two $S$-walks $S_i$ and $S_j$. If both $S_i$ and $S_j$ reverse directions, the primary diagonal of $q$ remains unchanged. If only one

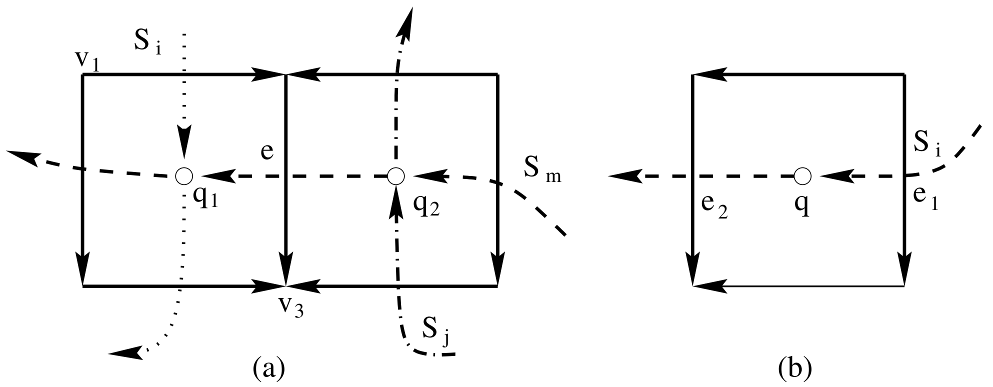


FIG. 2. (a) *An $S$-orientation $\mathcal{O}$ induces a $G$-orientation $\mathcal{G}$.* (b) *A $G$-orientation $\mathcal{G}$ derives an $S$-orientation $\mathcal{O}$.*

of the two $S$-walks reverses direction, then the primary and the secondary diagonals of $q$ swap.

DEFINITION 4. *Let $G$ be a 2MBP graph and let $\mathcal{G}$ be a $G$-orientation of $G$. We define an orientation of $G^*$ as follows. Consider any edge $e^*$ in $G^*$. Let $e$ be the edge in $G$ corresponding to $e^*$. When traveling $e$ along its direction in $\mathcal{G}$, we direct $e^*$ from the face $q_2$ on the left to the face $q_1$ on the right. This orientation of $G^*$ will be called the orientation* derived from $\mathcal{G}$ *and will be denoted by $\delta(\mathcal{G})$.*

LEMMA 2. *Let $G$ be a 2MBP graph. If $\mathcal{G}$ is a $G$-orientation $G$, then $\delta(\mathcal{G})$ is an $S$-orientation of $G^*$. If $\mathcal{G}, \mathcal{G}'$ form a $G$-orientation pair, then $\delta(\mathcal{G})$ and $\delta(\mathcal{G}')$ form an $S$-orientation pair.*

*Proof.* Let $\mathcal{S}(G^*) = \{S_1, S_2, \ldots, S_k\}$ be $S$-walks of $G^*$. Let $\mathcal{G}$ be a $G$-orientation of $G$. We want to show that every $S$-walk $S_i$ is a directed closed walk in the orientation $\mathcal{O} = \delta(\mathcal{G})$ of $G^*$. It is enough to show that any two consecutive edges on $S_i$ are assigned consistent directions in $\mathcal{O}$.

Consider any face $q$ on $S_i$. Let $e_1$ and $e_2$ be the two edges on the boundary of $q$ that are walked through by $S_i$. Note that $e_1$ and $e_2$ are opposite on the boundary of $q$. Let $e_1^*$ and $e_2^*$ be the dual edges corresponding to $e_1$ and $e_2$, respectively. Without lose of generality, assume $e_1^*$ is directed into $q$. We need to show $e_2^*$ is directed out of $q$ (see Figure 2(b)).

By the definition of $\delta(\mathcal{G})$, $e_1$ is clockwise on the boundary of $q$. Because $\mathcal{G}$ is a $G$-orientation, $e_1$ and $e_2$ must have different direction, and hence $e_2$ is directed counterclockwise on the boundary of $q$. By the definition of $\delta(\mathcal{G})$, $e_2^*$ is directed out of $q$ in $\mathcal{O}$, as will be shown. Since this is true for any face $q$ on $S_i$, $S_i$ is indeed a directed closed walk in $\mathcal{O}$. Hence $\mathcal{O}$ is an $S$-orientation of $G^*$.

Let $\mathcal{G}$ and $\mathcal{G}'$ be a $G$-orientation pair. For each edge $e$ of $G$, the direction of $e$ in $\mathcal{G}'$ is the reverse of that in $\mathcal{G}$. Hence, the direction of its dual edge $e^*$ in $\mathcal{O}' = \delta(\mathcal{G}')$ is the reverse of that in $\mathcal{O} = \delta(\mathcal{G})$. Therefore $\mathcal{O}$ and $\mathcal{O}'$ form an $S$-orientation pair of $G^*$. ☐

By Definitions 3 and 4, it's straightforward to verify that $\pi$ and $\delta$ are inverse mappings of each other. Hence we have the following.

THEOREM 4. *For a 2MBP $G$, the mapping $\pi$ (and $\delta$) is a one-to-one correspondence between the set of $G$-orientations of $G$ and the set of $S$-orientations of $G^*$, which preserves the pair relation.*

The following theorem describes how to obtain an even triangulation from a $G$-orientation of $G$.

THEOREM 5. *Let $G$ be a $2MBP$ graph.*

1. *Let $\mathcal{G}$ be a $G$-orientation of $G$. Then adding the primary diagonals into each face of $G$ results in an even triangulation of $G$, which is called the even triangulation* determined by $\mathcal{G}$, *denoted by $\mathcal{T}(\mathcal{G})$.*
2. *Let $\mathcal{G}$ and $\mathcal{G}'$ be two $G$-orientations of $G$. If $\mathcal{G}, \mathcal{G}'$ form a $G$-orientation pair of $G$, they determine the same even triangulation of $G$, i.e., $\mathcal{T}(\mathcal{G}) = \mathcal{T}(\mathcal{G}')$. If $\mathcal{G}, \mathcal{G}'$ do not form a $G$-orientation pair of $G$, they determine different even triangulations of $G$, i.e., $\mathcal{T}(\mathcal{G}) \neq \mathcal{T}(\mathcal{G}')$.*

*Proof.*

*Statement* 1. Let $G'$ be the graph resulting from adding the primary diagonals (with respect to $\mathcal{G}$) into the faces of $G$. Consider any vertex $v$ of $G$. Let $e_1, e_2, \dots, e_t$ be the edges in $G$ incident to $v$ in a clockwise direction. Thus $deg_G(v) = t$. Let $q_i$ $(1 \leq i \leq t)$ be the face incident to $v$ and with $e_i$ and $e_{i+1}$ on its boundary (where $e_{t+1} = e_1$).

We call $e_j$ a *go-in* (*go-out*, respectively) edge of $v$ if $e_j$ is directed into (out of, respectively) $v$ in $\mathcal{G}$. A face $q_i$ is called a *gap face* of $v$ if one of the two edges $e_i$ and $e_{i+1}$ is a go-in edge of $v$ and another is a go-out edge of $v$. $q_i$ is called a *good face* of $v$ if $e_i, e_{i+1}$ are both go-in edges of $v$ or both go-out edges of $v$. Denote the number of gap faces of $v$ by $gap(v)$. Since a gap face is always between a block of consecutive go-out edges and a block of consecutive go-in edges around $v$, it is easy to see that $gap(v)$ is always an even number.

The primary diagonal (with respect to $\mathcal{G}$) of each face $q$ is the diagonal connecting the starting and the ending vertices of the two directed paths on the boundary of $q$. Thus $v$ is incident to the primary diagonal of the face $q_i$ iff $q_i$ is a good face of $v$. Hence $deg_{G'}(v) = deg_G(v) + t - gap(v) = 2t - gap(v)$ is always even. Therefore $G' = \mathcal{T}(\mathcal{G})$ is an even triangulation of $G$.

Figure 3(a) shows an example of this construction. The solid lines are edges in $G$. The dotted lines are primary diagonals which are added into $G'$. The black dots indicate the gap faces. We have $deg_{G'}(v) = 2 \times 6 - 4 = 8$.

*Statement* 2. Suppose $\mathcal{G}$ and $\mathcal{G}'$ form a $G$-orientation pair of $G$. For any face $q$ of $G$, the primary diagonal of $q$ with respect to $\mathcal{G}$ is the same as that with respect to $\mathcal{G}'$. So $\mathcal{T}(\mathcal{G})$ is the same as $\mathcal{T}(\mathcal{G}')$.

Suppose $\mathcal{G}, \mathcal{G}'$ do not form a $G$-orientation pair. Then they have different sets of primary diagonals. So $\mathcal{T}(\mathcal{G})$ is different from $\mathcal{T}(\mathcal{G}')$. □

Note that the fact that the source and the sink of each face in a $G$-orientation are not adjacent plays a crucial role here. Only because of this, adding the primary diagonal (which connects the source and sink for each face) splits the quadrangle into two triangles. If the source and sink were adjacent, as possibly in the case of $st$-orientation, this would be impossible.

If $\mathcal{G}$ is induced from an $S$-orientation $\mathcal{O}$ of $G^*$, we also call $\mathcal{T}(\mathcal{G})$ the even triangulation *determined by* $\mathcal{O}$, and denote it by $\mathcal{T}(\mathcal{O})$.

Based on the discussion above, we have the following.

*New proof of Theorem* 1. Let $\mathcal{S}(G^*) = \{S_1, S_2, \dots, S_k\}$ be the $S$-walks of $G^*$. Arbitrarily assign a direction to each $S_i$. This gives an $S$-orientation $\mathcal{O}$ of $G^*$. By Lemma 1, we get a $G$-orientation $\mathcal{G} = \pi(\mathcal{O})$. By Theorem 5, we get an even triangulation $\mathcal{T}(\mathcal{G})$ of $G$. □

Theorem 5 states that every $G$-orientation pair determines an even triangula-

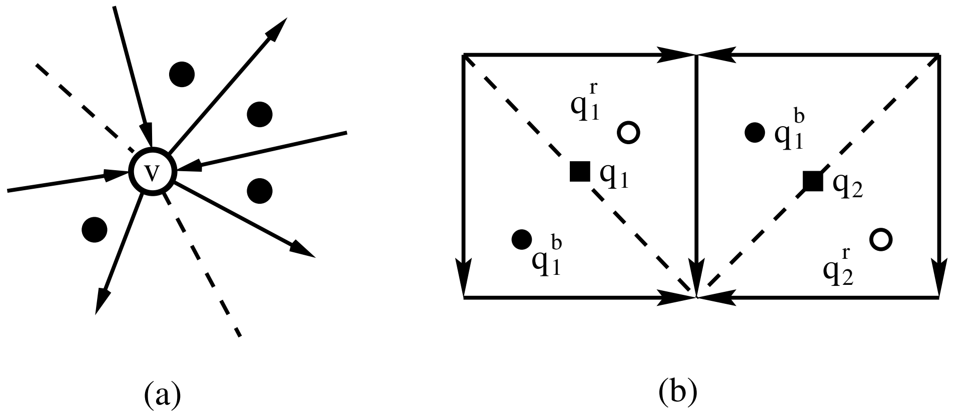


FIG. 3. (a) *Adding primary diagonals with respect to a G-orientation $\mathcal{G}$ results in even degree at vertex v.* (b) *An even triangulation of G induces a G-orientation $\mathcal{G}$ on faces $q_1$ and $q_2$.*

tion of $G$. Next we show that every even triangulation of $G$ is determined by a $G$-orientation pair of $G$.

THEOREM 6. *Let $G$ be a 2MBP graph and let $\mathcal{S}(G^*) = \{S_1, S_2, \ldots, S_k\}$ be the S-walks of $G^*$.*

1. *Every even triangulation $G'$ of $G$ is determined by a $G$-orientation pair of $G$.*
2. *$G$ has exactly $2^{k-1}$ distinct even triangulations.*

*Proof.*

*Statement* 1. Let $G'$ be any even triangulation of $G$. We want to show that there exists a $G$-orientation $\mathcal{G}$ of $G$ such that $G' = \mathcal{T}(\mathcal{G})$.

Let $G'^*$ be the dual graph of $G'$. For each face $q$ of $G$, the diagonal of $q$ from $G'$ splits $q$ into two faces, which will be called the *subfaces* of $G$. So $q$ contains two subfaces.

Since each vertex has even degree in $G'$, each facial cycle of $G'^*$ is of even length. Thus $G'^*$ is a bipartite plane graph. So we can color its vertices by two colors. In other words, we can color the subfaces of $G$ red and black so that no two red (black, respectively) subfaces are adjacent. Note that each face $q$ of $G$ contains one red and one black subface.

Consider any edge $e$ of $G$. Let $q_1$ and $q_2$ be the two faces of $G$ with $e$ on their common boundary. Let $q_1^r$ and $q_1^b$ be the red and the black subfaces contained in $q_1$, respectively. Let $q_2^r$ and $q_2^b$ be the red and the black subfaces contained in $q_2$, respectively. Note that exactly one red subface and one black subface from the four subfaces $q_1^r, q_1^b, q_2^r, q_2^b$ have $e$ on their boundaries. We direct $e$ clockwise on its neighboring red subface. (Hence, $e$ is directed counterclockwise on its black neighboring subface.) This way, each edge $e$ of $G$ is consistently assigned a direction. Let $\mathcal{G}$ denote this orientation of $G$.

Consider any face $q$ of $G$. Let $q^r$ and $q^b$ be the red and black subfaces contained in $q$, respectively. Two boundary edges of $q$ are the boundary edges of $q^r$. Hence they are directed clockwise and form a directed path of length 2 in $\mathcal{G}$. The other two boundary edges of $q$ are the boundary edges of $q^b$. Hence they are directed counterclockwise and form another directed path of length 2 in $\mathcal{G}$. Thus $\mathcal{G}$ is indeed a $G$-orientation of $G$.

Figure 3(b) shows an example of this construction. Two faces $q_1$ and $q_2$ are shown in the figure as black squares. Each of them contains two subfaces. The red subfaces are indicated by empty circles. The black subfaces are indicated by solid circles. The edges of $G$ are directed as described above.

For each face $q$ of $G$, the diagonal $d$ of $q$ from the even triangulation $G'$ is the diagonal which separates the red and the black subfaces of $q$. So it is the diagonal of $q$ connecting the starting and the ending vertices of the two directed paths on the boundary of $q$. Hence $d$ is the primary diagonal of $q$ with respect to $\mathcal{G}$. Therefore, the even triangulation $\mathcal{T}(\mathcal{G})$ determined by $\mathcal{G}$ is exactly $G'$.

*Statement* 2. It follows directly from the following facts:

- We can assign two directions to each $S$-walk $S_i$ $(1 \leq i \leq k)$. So $G^*$ has $2^k$ different $S$-orientations and $2^{k-1}$ different $S$-orientation pairs. By Theorem 4, there are exactly $2^{k-1}$ different $G$-orientation pairs of $G$.
- Each $G$-orientation pair determines a distinct even triangulation of $G$.
- Every even triangulation of $G$ is determined by a $G$-orientation pair. □

In order to provide a new proof of Theorem 3, we need the following.

LEMMA 3. *Let $G$ be a 2MBP and let $\mathcal{S}(G^*) = \{S_1, \dots, S_k\}$ be the $S$-walks of $G^*$. Let $\mathcal{O}_1$ be an $S$-orientation of $G^*$. Let $\mathcal{O}_2$ be the $S$-orientation of $G^*$ obtained from $\mathcal{O}_1$ by reversing the direction of an $S$-walk $S_i$.*

*Then $\mathcal{T}(\mathcal{O}_2)$ can be obtained from $\mathcal{T}(\mathcal{O}_1)$ by flipping the diagonals of all faces on $S_i$. (If $q$ occurs on $S_i$ twice, its diagonal is flipped twice and hence remains unchanged.)*

*Proof.* The proof directly follows from the following facts:

- If a face $q$ does not occur on $S_i$, its primary diagonal with respect to $\mathcal{O}_1$ is the same as that with respect to $\mathcal{O}_2$.
- If a face $q$ occurs on $S_i$ once, its primary diagonal with respect to $\mathcal{O}_1$ is its secondary diagonal with respect to $\mathcal{O}_2$.
- If a face $q$ occurs on $S_i$ twice, its primary diagonal with respect to $\mathcal{O}_1$ is the same as that with respect to $\mathcal{O}_2$. □

We are now ready to give the following.

*New proof of Theorem* 3.

*Statement* 1. Let $T$ be an even triangulation of a 2MBP $G$. Then $T$ is determined by an $S$-orientation $\mathcal{O}$ of $G^*$ by Theorem 6, i.e., $T = \mathcal{T}(\mathcal{O})$. Let $\mathcal{O}'$ be the $S$-orientation obtained from $\mathcal{O}$ by reversing the direction of an $S$-walk $S_i$. Let $T'$ be the even triangulation $\mathcal{T}(\mathcal{O}')$ of $G$ determined by $\mathcal{O}'$. By Lemma 3, $T'$ is obtained from $T$ by flipping the diagonals of the faces on $S_i$.

*Statement* 2. Let $T_1$ and $T_2$ be any two even triangulations of $G$. By Theorem 6, $T_1 = \mathcal{T}(\mathcal{O}_1)$ is determined by an $S$-orientation $\mathcal{O}_1$ and $T_2 = \mathcal{T}(\mathcal{O}_2)$ is determined by another $S$-orientation $\mathcal{O}_2$. Let $S_{i_1}, S_{i_2}, \dots, S_{i_t}$ $(t \leq k)$ be those $S$-walks whose directions in $\mathcal{O}_1$ and $\mathcal{O}_2$ are reversed. By repeated applications of Lemma 3, if we start with $T_1$ and flip the diagonals of the faces on the $S$-walks $S_{i_j}$ $(1 \leq j \leq t)$ one by one we get $T_2$. □

## 4. Algorithm.

Based on the discussion in section 3, we obtain the following:

ALGORITHM 1: EVEN TRIANGULATION.
INPUT: A 2-connected bipartite plane graph $G$.
OUTPUT: An even triangulation of $G$.

```
1. Add diagonals into $G$, if necessary, to make all facial cycles length 4. For
   simplicity, we still use $G$ to denote the resulting graph.
2. Construct the dual graph $G^*$.
```

3. Travel the edges of $G^*$ to trace the $S$-walks of $G^*$. Each $S$-walk is assigned a direction when being traveled.
4. For each face $q_i$ of $G$, identify its in-edges, out-edges, and the primary diagonal. Add the primary diagonal into $q_i$.

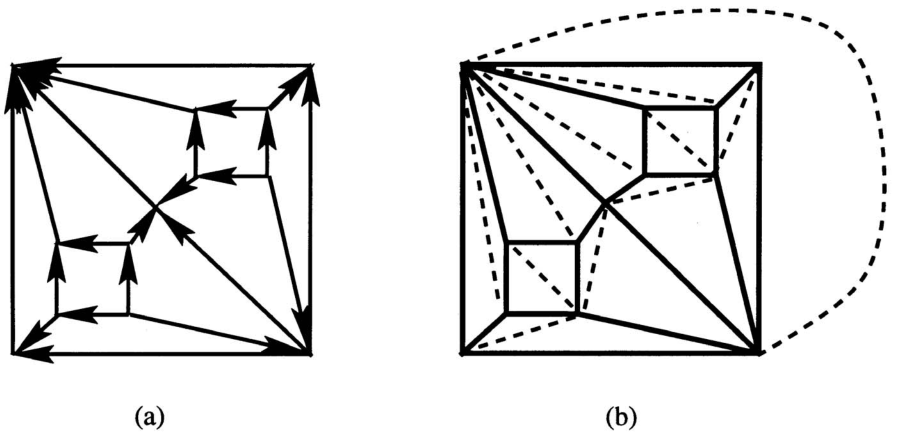


FIG. 4. (a) *A G-orientation $\mathcal{G}$ of $G$ corresponding to the S-orientation $\mathcal{O}$ in Figure* 1. (b) *An even triangulation of $G$ determined by $\mathcal{G}$.*

Figure 4 shows the even triangulation constructed by Algorithm 1 for the graph $G$ in Figure 1.

THEOREM 7. *Given a* 2*-connected bipartite plane graph $G$, Algorithm* 1 *finds an even triangulation of $G$ in $O(n)$ time.*

*Proof.* The correctness of Algorithm 1 directly follows from Theorem 5. All of the steps of Algorithm 1 can be easily implemented using elementary graph algorithm techniques (for example, see [2]), each in $O(n)$ time. □

Algorithm 1 may yield multiple edges as in the case of a quadrangle. It is impossible to obtain an even triangulation from a single quadrangle without allowing multiple edges.

## 5. Even triangulations of 2-connected graphs on high genus surfaces.

In this section, the term *surface* describes a closed, connected, orientable surface without boundary. Informally, it describes a sphere with $g$ ($g \geq 0$) handles where $g$ is the *genus* of the surface. We denote it by $\mathcal{F}_g$. A graph $G$ is said to be *embedded* on $\mathcal{F}_g$ if $G$ is drawn on $\mathcal{F}_g$ such that no two edges cross. The drawing of $G$ divides $\mathcal{F}_g$ into a number of connected regions. Each region is called a *face* of $G$. In this section, $G$ always denotes a graph that can be embedded on $\mathcal{F}_g$ and each of its faces is an open disc on $\mathcal{F}_g$. (Such a graph $G$ cannot be embedded on $\mathcal{F}_{g-1}$ without edge crossings. Although $G$ can be embedded on $\mathcal{F}_{g+1}$, some of its faces are not open discs on $\mathcal{F}_{g+1}$. See [6, 7].) Note that a plane graph is just a graph embedded on $\mathcal{F}_0$.

As for embedded planar graphs, an *even triangulation* $G'$ of $G$ is obtained from $G$ by adding diagonals into the faces of $G$ such that every face of $G'$ is a triangle and every vertex of $G'$ has even degree. In this section, we investigate the problem of determining the existence and the number of even triangulations of $G$. Note that, we also require the graph $G$ to be a multigraph without 2-cycle faces, which we will simply call graph $G$.

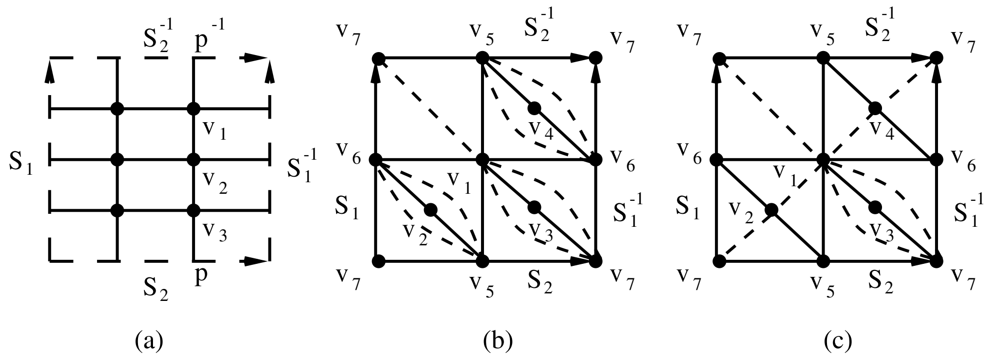


FIG. 5. (a) *A* 2*MEF graph* $G_1$ *on* $\mathcal{F}_1$ *with odd cycles.* (b) *An even triangulation of a* 2*MEF graph* $G$ *on* $\mathcal{F}_1$ *determined by its unique S-orientation pair.* (c) *An even triangulation of* $G$ *not determined by any S-orientation.*

If all of the facial cycles of $G$ are of even length, we call $G$ a 2-*connected even face graph on* $\mathcal{F}_g$ (or 2EF graph, for short). Obviously, if $G$ is a 2EF graph, we can add diagonals into its faces (if necessary) to make all of its facial cycles length 4. We call such a graph a 2-*connected maximal even face graph on* $\mathcal{F}_g$ (or 2MEF graph, for short). Note that 2MEF graphs are the counterpart of 2MBP graphs on high genus surfaces.

Our question is, given a 2MEF graph $G$, does $G$ always have an even triangulation? If so, how many?

According to Euler's formula, if $G$ is a 2MEF graph with $n$ vertices, $m$ edges, and $f$ faces, we have $n+f-m=2-2g$. Since each face of $G$ is a 4-cycle, we have $4f=2m$ and $f=n+2g-2$. The faces of $G$ will be denoted by $Q(G)=\{q_1,q_2,\ldots,q_{n+2g-2}\}$.

We can construct $\mathcal{F}_g$ from a $4g$ regular polygon on the plane (see [6, 7]). For example, to construct a torus $\mathcal{F}_1$ from a square (see Figure 5(a)), we glue the side $S_i$ with the side $S_i^{-1}$ ($i=1,2$) along their directions without twisting. The resulting surface is $\mathcal{F}_1$. Thus points $p$ and $p^{-1}$ in the figure become the same point on $\mathcal{F}_1$. The four corner points of the square become the same point on $\mathcal{F}_1$. In the figure, the line segment between point $v_1$ and point $p^{-1}$ and the line segment between point $p$ and point $v_3$ form a continuous line segment on $\mathcal{F}_1$ between $v_1$ and $v_3$. Figure 5(a) shows a 2MEF graph $G$ on $\mathcal{F}_1$ with 6 vertices (indicated by black dots), 12 edges (indicated by solid lines), and 6 faces. All the faces are 4-cycles. However, the three edges $(v_1,v_2),(v_2,v_3),(v_3,v_1)$ form a (nonfacial) cycle of length 3. So $G$ is *not* bipartite.

This example demonstrates a crucial difference between graphs on the plane and graphs on $\mathcal{F}_g$:

- If every face of a plane graph $G$ is an even cycle, then $G$ is bipartite.
- In contrast, even if every face of a graph $G$ embedded on $\mathcal{F}_g$ ($g>0$) is an even cycle, then $G$ is not necessarily bipartite.

As we will see, this difference makes it much harder to determine the number of distinct even triangulations of $G$.

Let $G$ be a 2MEF graph. Since $G$ is not necessarily bipartite, we cannot color the vertices of $G$ by two colors 0 and 1. So (1) does not make sense anymore. Hence we cannot rely on the existence of the solutions of (1) to show the existence of even triangulations of $G$. So the methods developed in [3, 4] cannot be used here at all.

On the other hand, most concepts and some theorems in sections 3 and 4 can be easily adapted to handle graphs $G$ on $\mathcal{F}_g$. Specifically, we can still construct the dual graph $G^*$ of $G$, where each vertex of $G^*$ is a face of $G$, and each edge of $G^*$ corresponds to an edge in $G$. Obviously $G^*$ can be embedded on $\mathcal{F}_g$. Moreover, since $\mathcal{F}_g$ is an orientable surface, it has two continuous normal vectors, and each remains consistent while traveling on $\mathcal{F}_g$. If we fix one normal vector of $\mathcal{F}_g$, we can still define the notions of clockwise, counterclockwise, right, and left directions on $\mathcal{F}_g$.

Thus the edge set of $G^*$ can still be partitioned into a set of $S$-walks. The concepts of 1-walk face, 2-walk face, $S$-orientation, $S$-orientation pair, $G$-orientation, $G$-orientation pair, the mappings $\pi$ and $\delta$, in-edge and out-edge, and primary diagonal can all be defined as in section 3 without any change. Moreover, the 2MEF graph versions of Lemmas 1 and 2 and Theorems 4 and 5 are still valid (whose proofs require no change). In particular, we state the following 2MEF graph version of Theorem 5.

THEOREM 8. *Let $G$ be a 2MEF graph on $\mathcal{F}_g$.*

1. *Let $\mathcal{G}$ be a $G$-orientation of $G$. Then adding the primary diagonals to each face of $G$ results in an even triangulation of $G$, which is called the even triangulation determined by $\mathcal{G}$, denoted by $\mathcal{T}(\mathcal{G})$.*
2. *Let $\mathcal{G}$ and $\mathcal{G}'$ be two $G$-orientations of $G$. If $\mathcal{G}, \mathcal{G}'$ form a $G$-orientation pair of $G$, they determine the same even triangulation of $G$, i.e., $\mathcal{T}(\mathcal{G}) = \mathcal{T}(\mathcal{G}')$. If $\mathcal{G}, \mathcal{G}'$ do not form a $G$-orientation pair of $G$, they determine different even triangulations of $G$, i.e., $\mathcal{T}(\mathcal{G}) \neq \mathcal{T}(\mathcal{G}')$.*

Because of Theorem 8, Algorithm 1 still correctly constructs an even triangulation of a 2EF graph $G$ in linear time. So the 2EF graph version of Theorem 7 is also valid.

Given any even triangulation $T$ of a 2MEF graph $G$, if we flip the diagonals of every face on an $S$-walk $S$ of $G^*$, we get another even triangulation. (If a face $q$ occurs twice on $S$, its diagonal is flipped twice and hence remains unchanged.) Thus statement 1 of Theorem 3 is still true.

The proof of statement 1 of Theorem 6 relies on the assumption that if all facial cycles of $G$ are even cycles, then $G$ is bipartite. Since this assumption is not true for 2MEF graphs, this proof is no longer valid. In fact, as we will show later, the two statements of Theorem 6 and statement 2 of Theorem 3 are *false* for 2MEF graphs.

We have the following weaker version of statement 2 of Theorem 6.

THEOREM 9. *Let $G$ be a 2MEF graph on $\mathcal{F}_g$ and let $\mathcal{S}(G^*) = \{S_1, \ldots, S_k\}$ be all $S$-walks of $G^*$. Then $G$ has at least $\max(2^{k-1}, 2^{2g-2})$ distinct even triangulations.*

*Proof.* Since we can assign two directions to each $S$-walk $S_i$ $(1 \le i \le k)$, $G^*$ has $2^{k-1}$ $S$-orientation pairs. Hence $G$ has $2^{k-1}$ distinct $G$-orientation pairs. By Theorem 8, $G$ has at least $2^{k-1}$ distinct even triangulations.

Now fix an even triangulation $T$ of $G$. We associate each face $q_i$ of $G$ with a $\{0,1\}$-valued variable $x_i$, and each triangulation $T'$ of $G$ with a vector $\vec{x} = (x_1, x_2, \ldots, x_{n+2g-2})$ in $GF(2)^{n+2g-2}$, where

$$T \text{ and } T' \text{ contain different diagonals of the face } q_i \Longleftrightarrow x_i = 1.$$

It is easy to see that (a) $T'$ is an even triangulation of $G$ iff its corresponding vector $\vec{x}$ is a solution of the following linear equation (2) over $GF(2)$; and (b) distinct even triangulations correspond to distinct solutions of (2) and vice versa:

$$\sum_{q_i \in Q_v} x_i = 0 \quad (\text{mod } 2) \quad (\forall v \in V). \tag{2}$$

Since (2) has $n$ equations and $n + 2g - 2$ variables, it has at least $2g - 2$ free variables. Thus, (2) has at least $2^{2g-2}$ distinct solutions (see [5]) and $G$ has at least $2^{2g-2}$ distinct even triangulations. ∎

Figures 5(b) and (c) show an example that the number of even triangulations of $G$ can be strictly bigger than $\max(2^{k-1}, 2^{2g-2})$. These figures show a 2MEF graph $G$ on $\mathcal{F}_1$ (after gluing the side $S_i$ with ${S_i}^{-1}$ for $i = 1, 2$). $G$ has 7 vertices (indicated by black dots and labeled $v_1, \ldots, v_7$) and 14 edges (indicated by solid lines). It can be verified that $G^*$ has a single $S$-walk on $\mathcal{F}_1$ and hence an unique $S$-orientation pair. Figures 5(b) and (c) show two distinct even triangulations of $G$ (where the added diagonals are indicated by dotted lines). The even triangulation shown in Figure 5(b) is determined by the unique $S$-orientation pair of $G^*$. So the even triangulation shown in Figure 5(c) is *not* determined by any $S$-orientation of $G^*$. Therefore statement 2 of Theorem 3 and all of Theorem 6 are not valid for a 2MEF graph $G$.

We raise the following open problem to conclude this paper: What properties of a 2MEF graph $G$ determine all its even triangulations and how many are they?

**Acknowledgment.** The authors would like to thank the anonymous referees for their helpful comments.

## REFERENCES

[1] J. A. Bondy and U. S. R. Murty, *Graph Theory with Applications*, North–Holland, New York, 1979.
[2] T. H. Cormen, C. E. Leiserson, and R. L. Rivest, *An Introduction to Algorithms*, McGraw–Hill, New York, 1990.
[3] F. Hoffmann and K. Kriegel, *A Graph-Coloring Result and Its Consequences for Polygon-Guarding Problems*, Technical Report TR-B-93-08, Inst. f. Informatik, Freie Universität, Berlin, 1993.
[4] F. Hoffmann and K. Kriegel, *A graph-coloring result and its consequences for polygon-guarding problems*, SIAM J. Discrete Math., 9 (1996), pp. 210–224.
[5] N. Jacobson, *Lectures in Abstract Algebras*, Springer-Verlag, New York, 1975.
[6] C. Kosniowski, *A First Course in Algebraic Topology*, Cambridge University Press, Cambridge, UK, 1980.
[7] J. Massey, *Algebraic Topology, An Introduction*, Harcourt, Brace and World, New York, 1967.
[8] P. Ossona de Mendez, *Orientations Bipolaires*, Ph.D. thesis, Ecole des Hautes Etudes en Sciences Sociales, Paris, 1994.
[9] R. J. Lipton, D. J. Rose, and R. E. Tarjan, *Generalized nested dissection*, SIAM J. Numer. Anal., 16 (1979), pp. 346–358.
[10] L. Lovász, *Combinatorial Problems and Exercises*, North–Holland, Amsterdam, 1979.
[11] J. O'Rourke, *Art Gallery Theorems and Algorithms*, Oxford University Press, New York, 1987.
[12] J. O'Rourke, *Computational geometry column* 15, SIGACT News, 23 (1992), pp. 26–28.
[13] P. Rosenstiehl and R. E. Tarjan, *Rectilinear planar layouts and bipolar orientations of planar graphs*, Discrete Comput. Geom., 1 (1986), pp. 343–353.

SIAM J. COMPUT.
Vol. 34, No. 3, pp. 697–719

# FAULT-TOLERANT SCHEDULING*

BALA KALYANASUNDARAM† AND KIRK R. PRUHS‡

**Abstract.** We study fault-tolerant multiprocessor scheduling under the realistic assumption that the occurrence of faults cannot be predicted. The goal in these problems is to minimize the delay incurred by the jobs. Since this is an online problem we use competitive analysis to evaluate possible algorithms. For the problems of minimizing the makespan and minimizing the average completion time (for static release times), we give nonclairvoyant algorithms (both deterministic and randomized) that have provably asymptotically optimal competitive ratios. The main tool used by these algorithms to combat faults is redundancy. We also show that randomization has the same effect as redundancy.

**Key words.** makespan, fault-tolerant, multiprocessor scheduling, competitive analysis, transient faults, permanent faults

**AMS subject classifications.** 68M15, 68Q99, 68W99

**DOI.** 10.1137/S0097539794261799

## 1. Introduction.

**1.1. Problem statement.** The scheduling of tasks in a multiprocessor system has been recognized as an important problem and has been extensively studied (see [6] for a survey). In a large multiprocessor system, processor faults are inevitable and fault-tolerance is a significant issue [11]. The vast majority of previous work on scheduling either assumes that there are no faults, or gives minimal analysis. In this paper we begin a theoretical investigation of the effect of processor faults on scheduling by considering several standard scheduling problems modified to allow faults. We assume that the pattern of faults cannot be predicted by the online scheduling algorithm. We then compare the schedules produced by the online algorithm to the schedules produced by an omniscient algorithm.

The setting for the generic multiprocessor scheduling problem is a collection $P_1, \ldots, P_m$ of processors. Each processor $P_j$ has a speed $s_j$. The processors are given a collection $J_1, \ldots, J_n$ of jobs. Each job $J_i$ has a *release time* $r_i$ that is the time that the online scheduling algorithm is first aware of $J_i$'s existence, and that is the earliest time that $J_i$ can begin execution. Furthermore, each $J_i$ has a length $x_i$, and running $J_i$ on $P_j$ takes $x_i/s_j$ units of time.

There are many variants of the multiprocessor scheduling problem depending on what assumptions one makes about the jobs and processors and on how one measures the desirability of a schedule. If all release times are 0, we say the release times are *static*. Otherwise, the release times are said to be *dynamic*. In the *identical processors case* all the processor speeds are equal. Otherwise, it is called the *related processors case*. A scheduling algorithm is *clairvoyant* if it learns each $x_i$ at time $r_i$,

*Received by the editors January 21, 1994; accepted for publication (in revised form) December 22, 2003; published electronically March 17, 2005.
http://www.siam.org/journals/sicomp/34-3/26179.html

†Department of Computer Science, Georgetown University, Washington, DC 20057 (kalyan@cs.georgetown.edu). The research of this author was supported in part by NSF under grants CCR-9202158, CCR-9734927, and CCR-0098271 and by Craves Family Professorship funds.

‡Department of Computer Science, University of Pittsburgh, Pittsburgh, PA (kirk@cs.pitt.edu). The research of this author was supported in part by NSF under grants CCR-9209283, CCR-9734927, and CCR-0098752.

and is *nonclairvoyant* if $x_i$ cannot be deduced until $J_i$ has been fully executed. A *preemptive* algorithm is allowed to suspend an executing job and later restart the job from the point of suspension. A *preemptive-with-restart* algorithm must begin the job afresh after suspending it. The *completion time* $c_i$ of a job $J_i$ is the time at which $J_i$ has been allocated enough time to finish execution. The *completion time* of $J_i$ is $w_i = c_i - r_i$. In this paper we primarily consider two scheduling measures, the *makespan* of a schedule, which is the maximum completion time, and the *average completion time*, which is $\frac{1}{n}\sum_{i=1}^{n} w_i$.

We assume that when a processor $P_j$ faults it is immediately detectable, and that the job currently being run on $P_j$ must be restarted from the beginning at some later point in time. A fault at $P_j$ can be classified as either *permanent*, in which case no more jobs may be run on $P_j$, or *transient*, in which case $P_j$ is inoperative for some finite period of time [11]. Note that a *nonclairvoyant* algorithm (also known as *online* algorithm) detects faults only at the time of their occurrence.

The *competitiveness* (or *competitive ratio*) of a deterministic algorithm for a particular problem and measure $\mathcal{M}$ is the supremum over all instances $\mathcal{I}$, of the ratio of the value of $\mathcal{M}$ for the schedule produced by the online algorithm given $\mathcal{I}$ as input to the optimal value of $\mathcal{M}$ for a schedule of $\mathcal{I}$ [9]. We can assume that the optimal value of $\mathcal{M}$ was computed by an offline algorithm with full advance knowledge of all the information about the jobs and the pattern of faults. For randomized algorithms we assume an oblivious adversary [2]; that is, the input (length of jobs and where, when, and how long the fault happens) must be specified in advance and may not be modified as a result of random events internal to the randomized algorithm. Thus the competitive ratio of a randomized algorithm on a particular problem and measure $\mathcal{M}$ is the supremum over all instances $\mathcal{I}$, of the ratio of the expected value of $\mathcal{M}$ for the schedule produced by the online algorithm given $\mathcal{I}$ as input to the optimal value of $\mathcal{M}$ for a schedule of $\mathcal{I}$.

The online algorithms that we give are nonclairvoyant; however, because our lower bound constructions all use equal length jobs, we can conclude that clairvoyance would not asymptotically help the online scheduling algorithm in the worst case. We always assume preemption-with-restart. As in most settings where fault-tolerance is an issue, the main tool available to combat faults is redundancy [11]. In this setting this generally means running multiple copies of the same job on different processors.

**1.2. Results.** The main results for the case of identical processors and preemption-with-restart are summarized in the table below. We use $\lambda$ to denote the total number of faults the scheduler will see over the entire length of its schedule. In the case of permanent faults we use $\eta = m - \lambda$ to denote the number of nonfaulty processors. The constant $\epsilon$ satisfies $0 < \epsilon < 1$. To determine the effect of the duration of faults on competitiveness, we assume that the duration of each transient fault is 0. That is, after a processor $P_j$ experiences a transient fault, a job may immediately be restarted on $P_j$. The results in this table suggest that for a moderate number (say, less than $\epsilon m$) of faults, the duration of the faults does not asymptotically affect the achievable competitive ratios. The function $\log^* x$ is defined to be the minimum $i$ such that $\log_2^{(i)} x \leq 1$, where $\log_2^{(i)}$ is the log function iterated $i$ times. The results for no faults come from [10, 8]. If one expects that permanent faults are largely independent, then the number of faults is likely to be approximately $\epsilon m$, where the constant $\epsilon$ is the probability that a particular processor faults. So in practice the most relevant line of the table is probably where $\lambda = \epsilon m$; note that the multiplicative constants here depend on $\epsilon$. These results show that the competitive ratios are effectively bounded in practice. The competitive ratio for average completion time is a constant, and the randomized competitive ratio for makespan is effectively bounded since $\log^* m$ is such a slowly growing function.

In the general related processors case, almost all the optimal competitive ratios listed in the table for permanent faults increase by a multiplicative factor of $R$, where $R$ is the ratio of the speed of the fastest processor to the speed of the slowest processor. For no faults, [10] showed a $\Theta(\min(\log R, \log m))$ bound on the optimal deterministic competitive ratio with respect to makespan, in the related processors case. In [8] it is implicitly shown that there exists a constant competitive algorithm for minimizing the average completion time in the general related processors case. The result is explicitly stated only for the identical processors case. This shows that as the system becomes more heterogeneous, the degradation of the optimal competitive ratio is much more rapid in the case of permanent faults.

Optimal Competitive Ratios for Identical Processors

| | Makespan | | Average completion time (static case) |
|---|---|---|---|
| Faults | Deterministic | Randomized | Deterministic and randomized |
| $\lambda = 0$ | $\Theta(1)$ [10] | $\Theta(1)$ [10] | $\Theta(1)$ [8] |
| $\lambda > 0$ Permanent | $\Theta(\max(\frac{\log m}{\log(\frac{m \log m}{\lambda})}, \frac{m}{\eta}))$ | $\Theta(\max(\log^* m - \log^* \frac{m}{\lambda}, \frac{m}{\eta}))$ | $\Theta(\frac{m}{\eta})$ |
| $\lambda = \epsilon m$ Permanent | $\Theta(\frac{\log m}{\log \log m})$ | $\Theta(\log^* m)$ | $\Theta(1)$ |
| $\lambda > 0$ Transient | $\Theta(\max(\frac{\log m}{\log(\frac{m \log m}{\lambda})}, \frac{\lambda}{m}))$ | $\Theta(\max(\log^* m - \log^* \frac{m}{\lambda}, \frac{\lambda}{m}))$ | $\Theta(\max(\frac{\lambda}{m}, 1))$ |

The results for makespan with permanent faults are given in section 2, and the results for average completion time with permanent faults are given in section 3. In section 4, we consider transient faults for the identical processors case. In section 5, we consider the effect of not allowing the online algorithm to run multiple copies of the same job at the same time. It is at least possible that this may be required in some situations where the programs have side effects or use external resources. We show that this restriction really cripples deterministic algorithms in that the optimal competitive ratio for both makespan and average completion time is $\Theta(\lambda)$ for permanent faults. We also show that the bounds on the competitive ratio, with respect to both makespan and average completion time, for randomized algorithms in the case of no redundancy are exactly the same as for deterministic algorithms that use redundancy.

We know of no previous theoretical investigations of this kind into fault-tolerant scheduling. Nonclairvoyant scheduling without faults is discussed in [5, 7, 8, 10]. In [8] it is shown that with dynamic release times and without faults, a nonclairvoyant deterministic (randomized) algorithm cannot be better than $\Omega(n^{1/3})$-competitive ($\Omega(\log n)$-competitive), with respect to average completion time, even allowing preemptions.

This shows that online scheduling algorithms face a much more daunting task when trying to minimize the average completion time of jobs with dynamic release times. For a general survey of fault-tolerant scheduling see [11].

## 2. Makespan.

**2.1. Lower bounds.** We use the following adversary to prove a lower bound on the competitive ratio, with respect to makespan, of any deterministic online algorithm. We define the *multiplicity* $\phi(J)$ of a job $J$ at some point in time to be the number of processors on which a copy of $J$ is being run at that time.

**2.1.1. Deterministic adversary.** *Adversary* DETERMINISTIC-DETER$(m, n, \lambda, \ell, \mathcal{A})$. Here $m$ is the number of unit speed processors, $\lambda$ is the number of faults, $n$ is the number of jobs, $\ell$ is the length of the jobs, and $\mathcal{A}$ is a deterministic online algorithm. We assume that $\ell$ is much larger than the length of a clock cycle. At time zero, the adversary presents $n$ jobs of length $\ell$ to $\mathcal{A}$. In the rest of the description, time is divided into stages, where the duration of each stage is $\ell$. Let $U(i)$, $1 \leq i \leq k$, be the collection of jobs that are not completed by $\mathcal{A}$ before the end of the $i$th stage. So $U(1)$ is the set of all $n$ jobs. We denote the cardinality of $U(i)$ by $u(i)$. We use $f(i)$ to denote the number of faults during the $i$th stage; we set the value of each $f(i)$ later. Intuitively, the goal of the adversary is to fault $f(i)$ working processors running jobs with lowest multiplicity just before the end of every stage $i$ (except possibly the last stage), so that the number of unfinished jobs at the end of the stage is maximized for $\mathcal{A}$. Observe that in order to force $\mathcal{A}$ not to finish a job $J$, the adversary must fault all of the machines running $J$.

We now show more formally how the adversary faults processors in each stage. Consider the multiplicity of jobs in $U(i)$ just before the end of the $i$th stage. Assume that the jobs in $U(i)$ are numbered and ordered such that $\phi(J_1) \leq \phi(J_2) \leq \cdots \leq \phi(J_{u(i)})$. Find the largest number $s$ such that $\sum_{j=1}^{s} \phi(J_j) \leq f(i)$. Let $U(i+1) = \{J_x : 1 \leq x \leq s\}$. Note that $s = u(i+1)$. Just before the end of the $i$th stage, the adversary faults every processor running a job in $U(i+1)$. Notice that since all the copies of jobs in $U(i+1)$ are terminated just before the end of the $i$th stage, by induction, no job in $U(i+1)$ can be completed before the end of the $(i+1)$st stage. Finally, let

$$D(i) = \begin{cases} U(i) - U(i+1) & \text{if } \sum_{j=1}^{s} \phi(J_j) = f(i), \\ U(i) - U(i+1) - \{J_{s+1}\} & \text{otherwise.} \end{cases}$$

We now explain how to set the value of the $f(i)$'s.

*Case* 1. If $m - \lambda \geq \frac{m \log \log m}{\log m}$, then each $f(i) = \lfloor \lambda / \log m \rfloor$ for $i \leq \log m$, and $f(i) = 0$ for $i > \log m$.

*Case* 2. If $m - \lambda < \frac{m \log \log m}{\log m}$, then $f(1) = f(2) = m/4$. In the permanent fault case, $f(3) = \lambda - m/2$, and $f(i) = 0$ for $i \geq 4$. In the transient fault case, $f(i) = m$ for $3 \leq i \leq 2 + \lfloor (\lambda - m)/m \rfloor$, and $f(i) = 0$ for $i > 2 + \lfloor (\lambda - m)/m \rfloor$.

**2.1.2. Analysis.** We first prove two simple lemmas that we will use often.

LEMMA 2.1. *Let $\alpha_1, \ldots, \alpha_k$ be a collection of nonnegative reals such that $\sum_1^k \alpha_i = \alpha$. Then there are at least $k/2$ integers $j$ such that $1 \leq j \leq k$ and $\alpha_j \leq 2\alpha/k$.*

*Proof.* To reach a contradiction, assume that for at most $k/2$ of the $j$'s we have $\alpha_j \leq 2\alpha/k$. Hence, there are at least $k/2$ $j$'s such that $\alpha_j > 2\alpha/k$. Therefore, the sum of those $\alpha_j$'s must be strictly larger than $(\frac{k}{2})(\frac{2\alpha}{k}) = \alpha$, a contradiction. ∎

LEMMA 2.2. *Let $g(i)$ be the number of working processors at the start of the $i$th stage. The adversary* DETERMINISTIC-DETER *guarantees that if $u(i) \geq \frac{g(i)}{c} + \frac{f(i)}{c-1}$, for some integer $c \geq 1$, then $u(i+1) \geq \lfloor \frac{f(i)}{c-1} \rfloor$.*

*Proof.* Consider some time just before the end of the $i$th stage. At this time there can be at most $g(i)/c$ jobs with multiplicity $c$ or greater. Hence, since $u(i) \geq \frac{g(i)}{c} + \frac{f(i)}{c-1}$, there must be at least $\lfloor \frac{f(i)}{c-1} \rfloor$ jobs with multiplicity $c-1$ or less. The result then follows by the definition of DETERMINISTIC-DETER. ◻

LEMMA 2.3. *Assume $\lambda \leq m - \frac{m \log \log m}{\log m}$ and $m \geq n \geq \lambda \geq 1$. Then for any online deterministic algorithm $\mathcal{A}$, the number of stages forced by the adversary* DETERMINISTIC-DETER$(m, n, \lambda, 1, \mathcal{A})$ *is $\Omega(\frac{\log m}{\log(\frac{m \log m}{\lambda})})$.*

*Proof.* Notice that the faults occur just before the end of every stage. We will assume that all $m$ processors are available to $\mathcal{A}$ at the beginning of every stage even if the faults are permanent. Therefore, the lower bound we will establish under this strong assumption holds for both permanent and transient fault cases. Notice that the required bound is constant if $\lambda \leq \sqrt{m}$. So it suffices to consider the case where $n \geq \lambda \geq \sqrt{m}$.

Consider the time just before the end of the $i$th stage. By the definition of $D(i)$, $m - \frac{\lambda}{\log m} \geq \sum_{J \in D(i)} \phi(J)$ and $|D(i)| \geq [u(i) - (1 + u(i+1))]$. The minimum multiplicity of a job in $D(i)$ is at least the maximum multiplicity of a job in $U(i+1)$, which in turn is at least $\frac{\lambda}{\log m} \cdot \frac{1}{1+u(i+1)}$. Therefore,

$$m - \frac{\lambda}{\log m} \;\geq\; \sum_{J \in D(i)} \phi(J) \;\geq\; [u(i) - (1 + u(i+1))] \, \frac{\lambda}{\log m} \, \frac{1}{1 + u(i+1)}.$$

Simplifying, we get $1+u(i+1) \geq u(i)(\frac{\lambda}{m \log m})$. By observing that $2u(i+1) \geq 1+u(i+1)$ for $u(i+1) \geq 1$, we get $u(i+1) \geq u(i)(\frac{\lambda}{2m \log m})$. Since $u(1) = n$, by induction we get $u(i+1) \geq n(\frac{\lambda}{2m \log m})^i$.

At the end of the last stage, say, $k$, we have $u(k+1) < 1$ and $u(k) \geq 1$. So

$$2 \geq 1 + u(k+1) \geq u(k) \left( \frac{\lambda}{m \log m} \right) \geq 2n \left( \frac{\lambda}{2m \log m} \right)^k.$$

Equivalently, we get $k \log(\frac{2m \log m}{\lambda}) \geq \log n$. Therefore, $k = \Omega(\frac{\log n}{\log(\frac{m \log m}{\lambda})})$. The result follows since $\log n \geq \frac{\log m}{2}$. ◻

LEMMA 2.4. *Assume that $\lambda$ satisfies $\lambda \geq m - \frac{m \log \log m}{\log m}$. Then for all deterministic online algorithms $\mathcal{A}$, the number of stages forced by the adversary* DETERMINISTIC-DETER$(m, m, \lambda, 1, \mathcal{A})$ *is $\Omega(\frac{m}{\eta})$ for the permanent fault case and $\Omega(\frac{\lambda}{m})$ for the transient fault case.*

*Proof.* Note that $f(1) = m/4$ and that there are $m$ working processors at the start of the first stage. Applying Lemma 2.2 with $c = 2$, $\mathcal{A}$ has at least $m/4$ jobs not completed by the start of the second stage. Note that $f(2) = m/4$ and that there are at most $m$ working processors at the start of the second stage. Applying Lemma 2.2 with $c = 6$, $\mathcal{A}$ has at least $m/20$ jobs not completed by the start of the third stage. Now consider the permanent fault case. Since there are exactly $\eta$ working processors at the end of the third stage and at least $m/20$ unfinished jobs before the end of the third stage, the number of stages is $\Omega(m/\eta)$. In the transient fault case, all processors are faulting for the next $\Omega(\lambda/m)$ stages. ◻

THEOREM 2.5. *In the case of identical processors and static release times, the competitive ratio, with respect to makespan, of any deterministic clairvoyant algorithm $\mathcal{A}$ is $\Omega(\max(\frac{\log m}{\log(\frac{m \log m}{\lambda})}, \frac{m}{\eta}))$ for $\lambda \geq 1$ permanent faults.*

*Proof.* Apply DETERMINISTIC-DETER$(m, m, \lambda, 1, \mathcal{A})$. Observe that the adversary faults a total of at most $m/2$ processors during the first two stages. As a consequence, at least $m/2$ processors do not experience a fault during the first two stages. These $m/2$ processors can complete $m$ jobs at the end of the second stage. Therefore, $OPT$, the makespan of the offline optimal algorithm, is at most 2. The result follows from Lemmas 2.3 and 2.4. ☐

THEOREM 2.6. *In the case of identical processors and static release times, the competitive ratio, with respect to makespan, of any deterministic clairvoyant algorithm $\mathcal{A}$ is $\Omega(\max(\frac{\log m}{\log(\frac{m \log m}{\lambda})}, \frac{\lambda}{m}))$ for $\lambda \geq 1$ transient faults.*

THEOREM 2.7. *In the case of related processors, the optimal deterministic competitive ratio, with respect to makespan, is then $\Omega(R \cdot \max(\frac{\log m}{\log(\frac{m \log m}{\lambda})}, \frac{m}{\eta}))$ for $\lambda \geq 1$ permanent faults.*

*Proof.* Let $\mathcal{A}$ be the given online algorithm. First we assume that $\lambda \leq \sqrt{m}$, where the lower bound we wish to establish is $\Omega(R)$. There are three processors with speed $R$ and $m - 3$ processors with speed 1. Initially, two jobs of length $R$ arrive. Just before the end of the first (unit time) stage, one speed $R$ processor faults so that there is an unfinished job at the end of the stage. At the end of the first stage, the two remaining speed $R$ processors fault. It is not hard to see that $OPT = 1$ and the online makespan is $\Omega(R)$.

We now assume that $\lambda > \sqrt{m}$. There are $\lambda/2$ processors with speed $R$ and $m - \lambda/2$ processors with speed 1. There are exactly $\lambda/4$ jobs of length $R$. Using Lemma 2.1, one can see that the adversary can fault $\lambda/4$ speed $R$ processors just before the end of the first stage such that at least $\lambda/16$ jobs are unfinished by the given online algorithm $\mathcal{A}$. The remaining speed $R$ processors fault right after time 1. Since $\lambda/4$ speed $R$ processors work for one full stage, $OPT = 1$. Now apply DETERMINISTIC-DETER$(m - \lambda/2, \lambda/16, \lambda/16, R, \mathcal{A})$. ☐

Note that in these lower bounds, the online algorithm is faced with a plethora of faults after the optimal schedule has finished, a phenomenon that may not be quite acceptable. It is not clear how to formulate an alternative reasonable model that avoids this phenomenon.

**2.1.3. Randomized case.** Using Yao's technique [3, 12], it suffices to prove the lower bound on the expected competitive ratio of any deterministic algorithm when faults occur according to a fixed probabilistic distribution. We first describe the instance of the input and the probability distribution on the faults. It is important to notice that the description of the following adversary is independent of the online algorithm.

*Adversary* OBLIVIOUS-DETER$(m, n, \lambda, \ell)$. Here $m$ is the number of unit speed processors, $\lambda$ is the number of faults, $n$ is the number of jobs, and $\ell$ is the length of the jobs. At time zero, the adversary releases $n$ jobs of length $\ell$. In the rest of the description, time is divided into stages, where the duration of each stage is $\ell$.

*Case* 1. Suppose $m - \lambda \geq \frac{m}{\log^* m}$. Just before the end of every stage (except possibly the last stage), each working processor will fault independently with probability $\frac{\lambda}{2m \log^* m}$.

*Case* 2. Suppose $m - \lambda < \frac{m}{\log^* m}$. Right before the end of the first two stages, each working processor faults independently with probability 1/5. In the permanent fault case, just before the end of the third stage, all but an arbitrary $m-\lambda$ processors fault. In the transient fault case, all processors fault just before the end of stages 3 through $2+\lfloor(\lambda-m)/m\rfloor$.

**2.1.4. Analysis.** Before analyzing this adversary we need the following definitions and facts.

DEFINITION 2.8. *For a nonnegative integer $k$, a real $\alpha \geq 2$, and a real $z \geq 1$, we inductively define*

$$\operatorname{tower}(\alpha,k,z) = \begin{cases} z & \text{if } k=0, \\ \alpha^{tower(\alpha,k-1,z)} & \text{otherwise.} \end{cases}$$

LEMMA 2.9. $(k-1)+\log^* \alpha^z \leq \log^*(\operatorname{tower}(\alpha,k,z)) \leq 2(k-1)+\log^* \alpha^z + O(1)$.

*Proof.* Notice that $2^{2^x} \geq \alpha^x$ if $x \geq \alpha \geq 2$. Therefore, for $k \geq 2$, we have

$$\operatorname{tower}(2,(k-1),\alpha^z) \leq \operatorname{tower}(\alpha,k,z) = \operatorname{tower}(\alpha,k-1,\alpha^z) \leq \operatorname{tower}(2,2(k-1),\alpha^z).$$

The result then follows since $\log^*(\operatorname{tower}(2,k,z)) = k+\log^* z + O(1)$. □

FACT 2.10. *If $p(x)$ is some fixed polynomially bounded function, then* $\log^* p(x) = \log^* x + O(1)$.

FACT 2.11. *Let $y$ be a random variable with expectation $\mu$ and variance $\sigma^2$. Then Chebyshev's inequality states that* $\operatorname{Prob}[|y-\mu| \geq \alpha\mu] \leq \frac{\sigma^2}{\alpha^2\mu^2}$. *If $\sigma^2 \leq \mu$, then* $\operatorname{Prob}[|y-\mu| \geq \alpha\mu] \leq \frac{1}{\alpha^2\mu}$.

LEMMA 2.12. *Assume $\lambda \leq m - \frac{m}{\log^* m}$ and $n = \Theta(m)$. Then for any deterministic online algorithm $\mathcal{A}$, the expected number of stages forced by the adversary* OBLIVIOUS-DETER$(m,n,\lambda,\ 1)$ *on $\mathcal{A}$ is $\Omega(\log^* m - \log^* \frac{m}{\lambda})$.*

*Proof.* In this proof we will assume that all $m$ processors are available to $\mathcal{A}$ at the beginning of every stage. Therefore, the desired bound holds for both permanent and transient fault cases. Notice that the required lower bound is $\Omega(1)$ if $\lambda \leq m/\log m$. So it suffices to consider the case where $\lambda > m/\log m$.

Consider a time just before the end of the $i$th stage. At this time let $u(i)$ be a random variable representing the number of jobs that are not completed by $\mathcal{A}$ before the end of the $i$th stage. We say that an unfinished job $J$ is *weak* if $\phi(J) \leq \frac{2m}{u(i)}$ at this time. Using Lemma 2.1, one can see that there are at least $u(i)/2$ *weak* jobs among these $u(i)$ jobs. We now show that with very high probability $u(i+1) \geq (\frac{u(i)}{4})(\frac{\lambda}{m\log^* m})^{\frac{2m}{u(i)}}$. Since processors fault independently with probability $\frac{\lambda}{2m\log^* m}$, the probability that no copy of a *weak* job $J$ is completed at the end of the $i$th stage is at least $(\frac{\lambda}{2m\log^* m})^{\phi(J)} \geq (\frac{\lambda}{2m\log^* m})^{\frac{2m}{u(i)}}$. Hence by linearity of expectation,

$$E[u(i+1)] \geq \left(\frac{u(i)}{2}\right)\cdot\left(\frac{\lambda}{2m\log^* m}\right)^{\frac{2m}{u(i)}}.$$

Since $u(i+1)$ is the sum of independent 0/1 valued random variables, the variance of $u(i+1)$ is at most the expected value of $u(i+1)$ (see [1]). So applying Chebyshev's inequality,

$$Pr[|u(i+1)-E[u(i+1)]| \geq E[u(i+1)]/2] \leq \frac{4}{E[u(i+1)]}.$$

We define a stage $i$ to be *good* if $|u(i+1)-E[u(i+1)]| \leq E[u(i+1)]/2$, and we define a stage $i$ to be *correct* if the number of faults during that stage does not exceed

$\lambda/\log^* m$. Assuming that all the stages are good and correct, we first show that there are $\Omega(\log^* m - \log^* \frac{m}{\lambda})$ stages. We then show that with probability $1-o(1)$ the first $\Theta(\log^* m - \log^* \frac{m}{\lambda})$ stages are good and correct.

If all stages are good as well as correct, then

$$u(i+1) \geq \frac{E[u(i+1)]}{2} \geq \left(\frac{u(i)}{4}\right) \cdot \left(\frac{\lambda}{2m\log^* m}\right)^{\frac{2m}{u(i)}}.$$

Let $r(i) = 2m/u(i)$; thus $r(1) = 2m/u(1) = 2m/n \geq 2$. Furthermore,

$$r(i+1) \leq 4r(i) \cdot \left(\frac{2m\log^* m}{\lambda}\right)^{r(i)}.$$

Since $r(i) \geq 2$, $\lambda \leq m$ and we have

$$r(i+1) \leq 2^{2+r(i)} \left(\frac{2m\log^* m}{\lambda}\right)^{r(i)} \leq \left(\frac{2m\log^* m}{\lambda}\right)^{3r(i)}.$$

Let $\alpha = (\frac{2m\log^* m}{\lambda})^3 \geq 8$. Therefore, we get $r(i+1) \leq \alpha^{r(i)}$, and so by induction we also get $r(i+1) \leq \text{tower}(\alpha, i, r(1))$.

Let $k$ be an integer such that $E[u(k)] > \sqrt{m}$ and $E[u(k+1)] \leq \sqrt{m}$. Our goal is to show that $k = \Omega(\log^* m - \log^* \frac{m}{\lambda})$. Since $|u(k+1) - E[u(k+1)]| \leq E[u(k+1)]/2$, we get $u(k+1) \leq 3\sqrt{m}/2$. As a consequence, $r(k+1) \geq 4\sqrt{m}/3$. Therefore, it must be the case that $\text{tower}(\alpha, k, r(1)) \geq 4\sqrt{m}/3$. Applying Lemma 2.9 we get that

$$\log^*(\text{tower}(\alpha, k, r(1))) \leq 2(k-1) + \log^* \alpha^{r(1)} + O(1) \quad \text{and}$$
$$\log^*(\text{tower}(\alpha, k, r(1))) \geq \log^*(4\sqrt{m}/3) \geq \log^* m.$$

Then by Fact 2.10

$$k \geq \frac{1}{2}(\log^* m - \log^* \alpha^{r(1)}) + O(1) = \frac{1}{2}\left(\log^* m - \log^* \frac{m\log^* m}{\lambda}\right) + O(1).$$

We now consider two cases. If $\lambda \leq m/\log^* m$, then $m/\lambda \geq \log^* m$. Thus

$$\log^* \frac{m\log^* m}{\lambda} \leq \log^* \left(\frac{m}{\lambda}\right)^2 = \log^* \frac{m}{\lambda} + O(1).$$

If $\lambda > m/\log^* m$, then $\log^* m - \log^* \frac{m\log^* m}{\lambda} = \Theta(\log^* m)$ and $\log^* m - \log^* \frac{m}{\lambda} = \Theta(\log^* m)$.

Let us now fix an $x$ such that $x \leq k$ and $x = \Theta(\log^* m - \log^* \frac{m}{\lambda})$. We now complete our proof by showing that the probability that the first $x$ stages are good and correct is $1-o(1)$. The probability that there exists a stage between 1 and $x$ that is not good is at most

$$\sum_{i=1}^{x} \frac{4}{E[u(i+1)]} \leq 4x/\sqrt{m}.$$

Hence, the probability that all the first $x$ stages are good is at least $1-o(1)$.

Since processors fault independently with probability $\lambda/2m\log^* m$, the expected number of faults per stage is at most $\lambda/2\log^* m$, and the variance does not exceed the expected number of faults. Applying Chebyshev's inequality, we get that the number of faults per stage is at most $\lambda/\log^* m$ with probability at least $1 - 2(\log^* m)/\lambda$. The probability that there is a stage between 1 and $x$ that is not correct is thus at most $2x(\log^* m)/\lambda$, which is $o(1)$ since $\lambda > m/\log m$. $\square$

We will use the following lemma repeatedly.

LEMMA 2.13. *Suppose that there are $n = \epsilon m$ unfinished unit length jobs and $m$ idle working processors at some time $t$. Let $\mathcal{A}$ be an assignment of these jobs to these processors. Further suppose that each processor faults independently with probability $p$ just before time $t+1$, where $p < 1$ is a constant. Then with probability $1 - O(1/m)$, the number of processors faulting just before time $t+1$ is at most $(1+\frac{1}{4})pm$. Also, with probability $1-O(1/m)$, the number of jobs unfinished before time $t+2$, using $\mathcal{A}$, is at least $\frac{\epsilon}{4}mp^{2/\epsilon} = \Theta(m)$.*

*Proof.* Since $p$ is a constant, $q = 1-p$ is also a constant. Let $x$ be the random variable representing the number of successes in $m$ Bernoulli trials, where the success probability is $p$. The expected number of successes $E(x)$ is $mp$ and the standard deviation $\sigma(x) = \sqrt{mpq} \le \sqrt{E(x)}$. Applying this to our case, the expected number of faults $E(x)$ is $mp$. Therefore, applying Chebyshev's inequality, the number of faults does not exceed $(1+\frac{1}{4})pm$ with probability $1 - 16/(p^2E(x)) = 1 - O(1/m)$.

Let $K$ be the set of jobs with multiplicity at most $2/\epsilon$ just before time $t+1$ in $\mathcal{A}$. Applying Lemma 2.1, we know that $|K| \ge n/2 = \epsilon m/2$. Since processors fault independently with probability $p$ just before time $t+1$, the probability that a job $J \in K$ is unfinished at time $t+1$ is $p^{2/\epsilon}$. Moreover, the probability that a job $J \in K$ is unfinished is independent of the probability that some other job is finished. Therefore, the expected number of jobs in $K$ that are not finished at time $t+1$ is at least $F = \frac{\epsilon}{2}mp^{2/\epsilon} = \Theta(m)$. Since the variance is smaller than the expected value, applying Chebyshev's inequality (see Fact 2.11), we get that with probability $1-4/F = 1-O(1/m)$, the number of unfinished jobs just before time $t+2$ is at least $F/2 \ge \frac{\epsilon}{4}mp^{2/\epsilon} = \Theta(m)$. ∎

LEMMA 2.14. *Assume $\lambda \ge m - \frac{m}{\log^* m}$ and $n = \Theta(m)$. Then for every deterministic online algorithm $\mathcal{A}$, the expected number of stages forced by the adversary* OBLIVIOUS-DETER$(m, n, \lambda, 1)$ *on $\mathcal{A}$ is $\Omega(\frac{m}{\eta})$ for the permanent fault case and $\Omega(\frac{\lambda}{m})$ for the transient fault case.*

*Proof.* Notice that the adversary (applying Case 2) faults each working processor independently with probability $1/5$ during the first two stages. So applying Lemma 2.13 twice, we get the following:

- With probability $1-O(1/m)$, the number of faults does not exceed $m/4$ during each of the first two stages.
- With probability $1-O(1/m)$, the number of jobs that cannot be finished before the end of the second stage is $\Omega(m)$.
- With probability $1-O(1/m)$, the number of jobs that cannot be completed before the end of the third stage is $\Omega(m)$.

The desired bound for the permanent fault case follows, since with probability $1-o(1)$, there are exactly $\eta$ working processors and $\Theta(m)$ unfinished jobs just before the end of the third stage. In the transient fault case, just before the end of each of the next $\lfloor(\lambda-m)/m\rfloor$ stages all $m$ processors fault. ∎

THEOREM 2.15. *In the case of identical processors and static release times, the competitive ratio, with respect to makespan, of any randomized clairvoyant algorithm $\mathcal{A}$ is $\Omega(\max(\log^* m - \log^* \frac{m}{\lambda}, \frac{m}{\eta}))$ for $\lambda \ge 1$ permanent faults.*

*Proof.* In the arguments of Lemmas 2.12 and 2.14, we also showed that at least $m/2$ processors are working during the first two stages with probability $1-o(1)$, and hence $OPT = 2$ for these cases. For other cases, $OPT$ is at most the makespan of $\mathcal{A}$. Therefore, $E[\frac{\text{makespan of } \mathcal{A}}{\text{makespan of } OPT}] = \Omega(\max(\log^* m - \log^* \frac{m}{\lambda}, \frac{m}{\eta}))$. Applying Yao's principle (see [3]), it is the case that the competitive ratio (against the oblivious adversary) of an online algorithm is not smaller than $E[\frac{\text{makespan of } \mathcal{A}}{\text{makespan of } OPT}]$. ∎

THEOREM 2.16. *In the case of identical processors and static release times, the competitive ratio, with respect to makespan, of any randomized clairvoyant algorithm $\mathcal{A}$ is $\Omega(\max(\log^* m - \log^* \frac{m}{\lambda}, \frac{\lambda}{m}))$ for $\lambda \geq 1$ transient faults.*

THEOREM 2.17. *In the case of related processors, the optimal randomized competitive ratio, with respect to makespan, is $\Omega(R)$ for $\lambda \geq 1$ permanent faults.*

*Proof.* Once again we use Yao's technique. There are three processors with speed $R$ and $m-3$ processors with speed 1. Initially, two jobs of length $R$ arrive. Just before the end of the first unit length stage, the adversary faults one of the three speed $R$ processors uniformly at random. As a consequence, with probability 1/3 there is an unfinished job at the end of the first stage. Now, at the end of the first stage, the remaining two speed $R$ processors fault. It is not hard to see that $OPT = 1$ and the expected makespan of any online algorithm is $\Omega(R)$. ∎

THEOREM 2.18. *In the case of related processors, the optimal randomized competitive ratio, with respect to makespan, is $\Omega(R \max(\log^* m, \frac{m}{\eta}))$ for $\lambda = \Theta(m)$ permanent faults.*

*Proof.* There are $\lambda/2$ processors with speed $R$ and $m - \lambda/2$ with speed 1. There are exactly $\lambda/4$ jobs of length $R$. Just before the end of the first stage, the adversary faults each of the speed $R$ processors independently with probability 1/3. Applying Lemma 2.13 we get the following:

- With probability $1-O(1/\lambda)$, the number of faults does not exceed $\lambda/4$ during the first stage.
- With probability $1 - O(1/\lambda)$, the number of jobs that cannot be finished before the end of the second stage is $\Omega(\lambda)$.

Since $\lambda = \Theta(m)$, with probability $1 - o(1)$ the number of unfinished jobs at the beginning of the second stage is $n = \Theta(\lambda)$ and at least $\lambda/4$ speed $R$ processors run without fault for one full stage; hence $OPT = 1$. At the end of the first stage, the adversary faults the remaining speed $R$ processors. Notice that $n \leq m$. Now the adversary applies the strategy OBLIVIOUS-DETER$(m-\lambda, \min(n, \lambda/2), \min(n, \lambda/2), R)$. Applying Lemmas 2.12 and 2.14, we get that with probability $1-o(1)$, the competitive ratio is $\Omega(R \cdot \max(\log^* m, \frac{m}{\eta}))$. Otherwise, the competitive ratio is at least 1. We get the desired result since the competitive ratio (against an oblivious adversary) is at least $E[\frac{\text{makespan of } \mathcal{A}}{\text{makespan of } OPT}]$. ∎

## 2.2. Upper bounds for makespan.

We define a randomized algorithm, which we call ROTARY, and analyze its deterministic worst-case performance and expected performance against permanent faults. We assume identical processors throughout this section.

DEFINITION 2.19. *The algorithm* ROTARY *breaks time into stages.*

1. *Let $S_i$ denote the ith stage.*
2. *Let $u(i)$ denote the number of unfinished jobs at the beginning of $S_i$.*
3. *Let $g(i)$ denote the number of working processors at the beginning of $S_i$.*
4. *Let $c(i)$ denote the number of distinct jobs completed during $S_i$.*
5. *Let $f(i)$ denote the number of faults experienced during $S_i$.*
6. *Let $OPT$ be the optimal makespan.*

### 2.2.1. Online algorithm.

ALGORITHM ROTARY. The algorithm divides time into stages. The first stage starts when the first job is released. We maintain the invariant that all working processors are idle at the start of a stage. At the beginning of a stage $S_i$, the algorithm initializes a queue $Q_i$ to contain the collection of jobs that have been released but not finished by the start of $S_i$. The ordering of the jobs in $Q_i$ is arbitrary. The rest of the execution of ROTARY depends on the relationship between $u(i)$ and $g(i)$.

Suppose $u(i) > g(i)$. Whenever a processor $P$ is idle, $P$ removes a job at the front of the queue $Q_i$ and runs it. If $Q_i$ is empty, $P$ remains idle for the rest of this stage. Stage $S_i$ ends at the earliest time such that every job in $Q_i$ has run to completion or has experienced a fault.

Now, suppose $u(i) \leq g(i)$. The algorithm selects uniformly at random an ordering of the $g(i)$ working processors and then assigns jobs to processors in this order. When considering a processor $P$, Rotary assigns $P$ the job $J$ currently at the front of $Q_i$, and then moves $J$ to the back of $Q_i$. No processor is assigned a second job. Stage $S_i$ ends at the earliest time such that every processor has either finished a job or has faulted.

**2.2.2. Basic properties of Rotary.** Note that Rotary often forces a processor to remain idle for the remainder of a stage after having completed a job. This enables us to simplify many of our definitions and proofs. The asymptotic bounds do not change if we allow some arbitrary collection of idle processors to run any unfinished job. Observe that when a new job arrives in the middle of a stage, it will not be added to the queue. It will be considered only during the following stages.

Our first goal is to show that by some reasonable time Rotary has reduced the number of unfinished jobs to $m$.

Fact 2.20. *For all jobs $J_i$, $x_i \leq OPT$.*

Lemma 2.21. Rotary *guarantees that the length of any stage $i$ is at most $(\frac{m}{\eta} + 1)OPT$.*

*Proof.* If $u(i) \leq g(i)$, then, applying Fact 2.20, the length of the stage is at most $OPT$. Suppose $u(i) > g(i)$. Since there are always at least $\eta$ fault-free processors, a simple counting argument shows that for every job $J$ in the queue, some processor must start executing a copy of $J$ by at least time $m \cdot OPT/\eta$ after the start of the stage. Using Fact 2.20, our claim follows. ▯

Lemma 2.22. Rotary *guarantees that there will be at most $m$ jobs left unfinished $(2\frac{m}{\eta} + 2)OPT$ time units after the release of the last job.*

*Proof.* By Lemma 2.21, a new stage $S$ starts no later than $(\frac{m}{\eta} + 1)OPT$ time units after the release of the last job. Since there are at most $m$ faults in stage $S$, at most $m$ jobs are left unfinished after stage $S$, which lasts at most $(\frac{m}{\eta} + 1)OPT$ units of time. ▯

Now we focus our attention on the case where the number of unfinished jobs is at most $m$. Intuitively, we will argue that for a stage in which the multiplicity of a job is not too small and not too many processors fault, the online algorithm makes good progress. On the other hand, we will also argue that there cannot be too many stages with either small multiplicity or too many faults.

Definition 2.23.

1. *Let $\phi(i)$ denote the minimum, over all unfinished jobs $J$ at the start of stage $S_i$, of the number of processors that ran a copy of $J$ in stage $i$.*
2. *Let stage $S_\beta$ be the first stage, after the release of the last job, for which the number of jobs left unfinished at the start of this stage is at most $m$.*
3. *Let $C = \{S_i : i \geq \beta\}$.*
4. *Let $F = \{S_i \in C : \phi(i) \leq 2\}$. $F$ is the set of all stages in $C$ such that the algorithm ran at most two copies of some unfinished job during that stage.*
5. *Let $H = \{S_i \in C - F : f(i) \leq g(i)/4\}$. $H$ is the set of all stages in $C - F$ such that the ratio of the number of working processors to the number of processors experiencing a fault is at least 4.*

LEMMA 2.24. ROTARY *guarantees that the number of copies of any unfinished jobs that are started during $S_i$ is either $\phi(i)$ or $\phi(i)+1$.*

Since $\eta$ processors never experience a fault, for any stage in $F$, at least $\lceil \eta/3 \rceil$ jobs will be completed. Therefore $|F| \le 3\lceil \frac{m}{\eta} \rceil$. On the other hand, we will derive a bound on $|C-F-H|$, since the number of faults in each stage in $C-F-H$ is large.

LEMMA 2.25. ROTARY *guarantees that if the length of a stage $S_i \in F$ is $\alpha \cdot OPT$, then $\lfloor \alpha - 1 \rfloor \le 3c(i)/\eta$.*

*Proof.* Assume that stage $S_i$ finished at time $t$. The last job assignment in this stage cannot be made earlier than time $t - OPT$ by Fact 2.20. Observe that any processor that does not experience a fault during a stage cannot be idle for more than $OPT$ time units. Thus the $\eta$ nonfaulty processors were all running jobs at all times before $t - OPT$, and thus by Fact 2.20 each of these processors must complete at least $\lfloor \alpha - 1 \rfloor$ jobs. Since $S_i \in F$, $\phi(i) \le 2$ and so by applying Lemma 2.24 we conclude that $c(i) \ge \eta \lfloor \alpha - 1 \rfloor / 3$. □

LEMMA 2.26. ROTARY *guarantees that the total length of the stages in $F$ is at most $9\lceil \frac{m}{\eta} \rceil$.*

*Proof.* First observe that $\eta$ nonfaulty processors complete a job in each stage. For a stage in $F$, at most three processors finish the same job. Since the number of jobs in any of the stages in $C$ is at most $m$, $|F| \le 3\lceil \frac{m}{\eta} \rceil$. Let $\alpha_i OPT$ be the length of stage $S_i \in F$. Consider the sum

$$\sum_{S_i \in F} \alpha_i OPT \le OPT \cdot \sum_{S_i \in F} (\lfloor \alpha_i - 1 \rfloor + 2).$$

Applying Lemma 2.25, this sum is at most $OPT \cdot (2|F| + 3\sum_{S_i \in F} c(i)/\eta)$. Applying the bound on $|F|$, the result follows since $\sum_{S_i \in F} c(i) \le m$. □

LEMMA 2.27. ROTARY *guarantees that the length of a stage in $C-F$ is at most $OPT$.*

*Proof.* Suppose $S_i \in C-F$. By the definition of $F$, $\phi(i) > 3$. This can happen only when $u(i) \le g(i)$. Under this situation, according to the description of ROTARY, no processor is assigned a second job during this stage. Applying Fact 2.20, the result follows. □

LEMMA 2.28. ROTARY *guarantees that $|C-F-H|$ is at most $\log_{4/3} \frac{m}{\eta}$.*

*Proof.* Observe that faults are permanent, and that initially there are $m$ working processors. Also, for any stage $S_i \in C-F-H$, $f(i) \ge \frac{1}{4} g(i)$. As a consequence, after $y$ stages in $C-F-H$, the number of remaining working processors is at most $m/(3/4)^y$. The result then follows, since $\eta$ processors never fault. □

LEMMA 2.29. *The cumulative length of the stages not in $H$ is $O(\frac{m}{\eta} OPT)$.*

*Proof.* The result follows from Lemmas 2.22, 2.26, 2.27, and 2.28. □

In order to prove the desired upper bound for ROTARY, it now suffices to prove that the cumulative length of the stages in $H$ is not large. Since $H \subseteq C-F$, by Lemma 2.27 it suffices to show an upper bound on the number of stages in $H$.

**2.2.3. Deterministic bound.** In this subsection, we will analyze the worst-case deterministic performance of ROTARY. We assume that the ordering of the processors in ROTARY is determined by some arbitrary deterministic procedure. We call the resulting algorithm DETERMINISTIC ROTARY.

DEFINITION 2.30. *For $1 \le j \le \lfloor 1 + \log \frac{m}{\eta} \rfloor$,*

1. *let $N_j = \{S_i \in H : m/2^j < g(i) \le m/2^{j-1}\}$;*
2. *let $\lambda_j$ be the total number of faults experienced during stages in $N_j$, i.e., $\lambda_j = \sum_{S_i \in N_j} f(i)$;*

3. *let* $n_j = |N_j|$.

LEMMA 2.31. ROTARY *guarantees that* $\lambda_j \le m/2^{j-1}$.

*Proof.* This follows since faults are permanent and $g(i) \le m/2^{j-1}$ for any stage $S_i \in N_j$. ∎

LEMMA 2.32. DETERMINISTIC ROTARY *guarantees that the cardinality of* $H$ *is* $O(\max(\frac{\log m}{\log(\frac{m \log m}{\lambda})}, \frac{m}{\eta}))$.

*Proof.* We say that a stage $S_i \in N_j$ is *good* if $f(i) \le 2\lambda_j/n_j$. By Lemma 2.1 there can be at most $n_j/2$ stages in $N_j$ that are not good. Hence, at least $\frac{1}{2}|H|$ stages in $H$ are good. From now on, we consider only good stages in $H$. Let $k$ be the total number of good stages in $H$.

Consider a good stage $S_i \in N_j$. The algorithm guarantees that $\phi(i) \ge \lfloor g(i)/u(i) \rfloor$. Notice that $H \cap F = \emptyset$. So $S_i \notin F$, and as a consequence $2\phi(i) \ge \phi(i)+1 \ge g(i)/u(i)$. Since $m/2^j < g(i) \le m/2^{j-1}$, we get $\phi(i) \ge m/(2^{j+1}u(i))$.

Since the multiplicity of a job is at least $\phi(i)$, it must be the case that $\phi(i)u(i+1) \le f(i)$. Since $S_i$ is good, $f(i) \le 2\lambda_j/n_j$ and so $u(i+1) \le (\frac{2\lambda_j}{n_j})/\phi(i)$. Now substituting $\phi(i) \ge m/(2^{j+1}u(i))$ into this inequality we get

$$u(i+1) \le u(i)\left(\frac{2\lambda_j}{n_j}\right)\left(\frac{2^{j+1}}{m}\right) = u(i)8\left(\frac{\lambda_j}{n_j}\right)\left(\frac{2^{j-1}}{m}\right). \tag{1}$$

Let $\ell = \lfloor 1 + \log_2 \frac{m}{\eta} \rfloor$. We now measure the progress made by good stages in $H$. We set $u'(i)$ to be equal to $u(x)$, where $S_x$ is the chronologically $i$th good stage. Applying Definition 2.23, the number of unfinished jobs in any stage in $H \subset C$ is at most $m$. Therefore, we have $u'(1) \le m$. Since at least $n_j/2$ stages in $N_j$ are good, applying induction we get

$$u'(k) \le m8^k \prod_{j=1}^{\ell} \left(\frac{\lambda_j}{n_j}\frac{2^{j-1}}{m}\right)^{n_j/2}. \tag{2}$$

Recall that $k$ is the number of good stages in $H$. First assume that $\eta > m/2$. Notice that then $\ell = 1$, $\lambda_1 \le \lambda$, and $k \le n_1 \le 2k$. Observe that $u'(k) \ge 1$; otherwise there will be at most $k-1$ stages. Then (2) implies

$$m8^k\left(\frac{\lambda}{n_1 m}\right)^{k/2} \ge 1 \quad \text{or} \quad \log m \ge \frac{k}{2}\log\left(\frac{n_1 m}{2\lambda}\right) - 3k \ge \frac{k}{2}\log\left(\frac{km}{2\lambda}\right) - 3k.$$

Therefore, $k = O(\frac{\log m}{\log(\frac{k \log k}{\lambda})})$.

We now consider the case that $m/2 \ge \eta \ge m/\log m$. Call an $n_j$ *big* if $n_j \ge k/(2\ell)$. Observe that by Lemma 2.31, $\lambda_j \le \frac{m}{2^{j-1}}$. Notice that the $u'(i)$'s do not increase as we move from one stage to another. As before, applying the fact that $u'(k) \ge 1$, we will show that $k$ is not large. In order to estimate $k$, we consider the rate at which the $u'(i)$'s decrease during a good stage where $n_i$ is *big*. So, applying $u'(k) \ge 1$, we can conclude that

$$m8^k \prod_{\text{big } n_j}\left(\frac{2\ell}{k}\right)^{n_j/2} \ge m8^k \prod_{\text{big } n_j}\left(\frac{\lambda_j}{n_j}\frac{2^{j-1}}{m}\right)^{n_j/2} \ge m8^k \prod_{j=1}^{\ell}\left(\frac{\lambda_j}{n_j}\frac{2^{j-1}}{m}\right)^{n_j/2} \ge 1. \tag{3}$$

In the above expression, the first and the second inequalities follow since $n_j \geq k/(2\ell)$ and for any $j$ in $[1, \ell]$, $\frac{\lambda_j}{n_j}\frac{2^{j-1}}{m} \leq 1$. Notice we can use Lemma 2.1 to show that $\sum_{\text{big } n_j} n_j \geq k/2$. Therefore, (3) implies

$$m8^k \prod_{\text{big } n_j} \left(\frac{2\ell}{k}\right)^{n_j/2} = m8^k \left(\frac{2\ell}{k}\right)^{\sum_{\text{big } n_j} n_j/2} \leq m8^k \left(\frac{2\ell}{k}\right)^{k/4}. \tag{4}$$

As a consequence, we get

$$m8^k \left(\frac{2\ell}{k}\right)^{k/4} \geq 1 \quad \text{or} \quad \frac{k}{4}\log\frac{k}{4\cdot 8^4\ell} \leq \log m. \tag{5}$$

Then by the definition of $\ell$, $\ell = O(\log\log m)$. Then (5) implies $k = O(\log m/\log\log m)$. Note that $\log m/\log\log m = O(\frac{\log m}{\log(\frac{m\log m}{\lambda})})$ if $m/2 \geq \eta \geq m/\log m$.

Now assume $\eta < m/\log m$. Since $\ell = O(k)$, statement (5) implies $k = O(\log m)$. Note that $\log m = O(\frac{m}{\eta})$ for $\eta < m/\log m$. The result follows since $k \geq \frac{1}{2}|H|$. ∎

THEOREM 2.33. *In the case of permanent faults and identical processors, the competitive ratio of* DETERMINISTIC ROTARY *is* $O(\max(\frac{\log m}{\log(\frac{m\log m}{\lambda})}, \frac{m}{\eta}))$ *for makespan.*

*Proof.* The result follows from Lemmas 2.29, 2.32, and 2.27. ∎

**2.2.4. Randomized bound.** We now examine the expected performance of the randomized algorithm ROTARY.

LEMMA 2.34. *The expected number of stages in $H$ for* ROTARY *is* $O(\log^* m - \log^* \frac{m}{\lambda})$.

*Proof.* In the following proof, we will not use the fact that $\lambda \leq m$. As a consequence, this lemma is also true for the transient faults case where $\lambda$ could be significantly larger than $m$. We will appeal to this fact when we discuss transient faults.

Let $S_i$ be a stage in $H$. For $1 \leq j \leq \phi(i)$, let $P_j(i)$ be the set of processors that were assigned a job from the $j$th time through the queue in $S_i$; e.g., the first $u(i)$ processors assigned jobs are in $P_1(i)$. Let $f_j(i)$ be a random variable representing the number of faults experienced by processors in $P_j(i)$ during stage $S_i$. Recall that $f(i)$ is the total number of faults during stage $S_i$.

For $1 \leq a \leq u(i)$, let $x_a$ be a random variable equal to 1 if all copies of the $a$th job starting $S_i$ experience a fault during stage $S_i$, and 0 otherwise. Observe that processors are randomly ordered before they start selecting a job from the queue. As a consequence, the probability that a job $J_a$ is not completed by a processor that selected $J_a$ during the $j$th time through the queue is $\frac{f_j(i)}{u(i)}$. Then

$$E[x_a] = Pr[J_a \text{ not finished during } S_i] \leq \prod_{i=1}^{\phi(i)} \frac{f_j(i)}{u(i)}. \tag{6}$$

For a fixed number $\delta$ of faults during the stage $S_i$ and $a \neq b$, notice that

$$Pr[x_a = 1 \mid x_b = 1,\, f(i) = \delta] \leq Pr[x_a = 1 \mid f(i) = \delta].$$

Therefore, we can conclude that

$$Pr[x_a = 1,\, x_b = 1 \mid f(i) = \delta] \leq Pr[x_a = 1 \mid f(i) = \delta] \cdot Pr[x_b = 1 \mid f(i) = \delta].$$

Since the $x_i$'s are 0/1-random variables, $\text{Cov}[x_a, x_b] = Pr[x_a = 1, x_b = 1] - Pr[x_a = 1] \cdot Pr[x_b = 1]$. As a consequence, for $a \neq b$, the covariance $\text{Cov}[x_a, x_b] \leq 0$. For 0/1-random variables $x_a$ (see [1]),

$$\text{(7)} \qquad \text{Var}\left[\sum_{j=1}^{u(i)} x_j\right] \leq E\left[\sum_{j=1}^{u(i)} x_j\right] + \sum_{a \neq b} \text{Cov}[x_a, x_b].$$

As a consequence, we get

$$\text{(8)} \qquad \text{Var}\left[\sum_{j=1}^{j=u(i)} x_j\right] \leq E\left[\sum_{j=1}^{j=u(i)} x_j\right].$$

The product in (6) is maximum subject to $\sum_{j=1}^{\phi(i)} f_j(i) \leq f(i)$ if each $f_j(i) = f(i)/\phi(i)$. Hence for $1 \leq a \leq u(i)$,

$$\text{(9)} \qquad E[x_a] \leq \prod_{i=1}^{\phi(i)} \frac{f(i)}{u(i)\phi(i)}.$$

Note $u(i)\phi(i) \geq u(i)(\frac{g(i)}{u(i)} - 1)$. Since $S_i \notin F$, it is the case that $u(i) \leq g(i)/2$ and $u(i)(\frac{g(i)}{u(i)} - 1) \geq g(i)/2$. Hence,

$$\text{(10)} \qquad E[x_a] \leq \prod_{i=1}^{\phi(i)} \frac{2f(i)}{g(i)}.$$

Let $k$ be the number of stages in $H$, and let $\alpha$ be the number of faults experienced during stages in $H$. Since $f(i) \leq g(i)/4$ for any $S_i$ in $H$, we have $2f(i)/g(i) \leq 1/2$. Call a stage $S_i \in H$ *good* if $f(i) \leq \frac{2\alpha}{k}$. Using Lemma 2.1 we can show that at least $\frac{1}{2}|H|$ stages in $H$ are good. So, it suffices to find an upper bound on the expected number of good stages in $H$. Let $S_i$ be a good stage under consideration. Hence,

$$\text{(11)} \qquad E[x_a] \leq \left(\min\left(\frac{1}{2}, \frac{4\alpha}{g(i)k}\right)\right)^{\phi(i)} \leq \left(\min\left(\frac{1}{2}, \frac{4\lambda}{g(i)k}\right)\right)^{\phi(i)}$$

and, applying linearity of expectation, we get

$$\text{(12)} \qquad E\left[\sum_{j=1}^{u(i)} x_j\right] = E[u(i+1)] = \sum_{j=1}^{u(i)} E[x_j] \leq u(i)\left(\min\left(\frac{1}{2}, \frac{8\lambda}{g(i)k}\right)\right)^{\phi(i)}.$$

Now, applying (8), we get

$$\text{(13)} \qquad \text{Var}[u(i+1)] \leq u(i)\left(\min\left(\frac{1}{2}, \frac{8\lambda}{g(i)k}\right)\right)^{\phi(i)}.$$

Call a stage $S_i$ in $H$ *unlucky* if $u(i+1) > 3E[u(i+1)]$. Since

$$3u(i)\left(\min\left(\frac{1}{2}, \frac{8\lambda}{g(i)k}\right)\right)^{\phi(i)} \geq 3E[u(i+1)],$$

we have

$$u(i+1) \leq 3u(i)\left(\min\left(\frac{1}{2}, \frac{8\lambda}{g(i)k}\right)\right)^{\phi(i)}$$

for any *lucky* stage. By Chebyshev's inequality, the probability that a stage is *unlucky* is at most $\mathrm{VAR}[u(i+1)]/(2E[u(i+1)])^2 \leq 1/4$. Since at least $k/2$ stages in $H$ are good, with probability $1 - O(1/k)$, at least $k/4$ good stages in $H$ are *lucky*.

Let $k' = \Theta(k)$ be the number of *lucky* stages in $H$. We ignore the progress made by ROTARY during stages in $H$ that are *unlucky*. Suppose stage $x$ is *lucky*. Then

$$u(x+1) \leq 3u(x)\left(\min\left(\frac{1}{2}, \frac{8\lambda}{g(x)k}\right)\right)^{\phi(x)} \leq 3u(x)\left(\min\left(\frac{1}{2}, \frac{8\lambda}{g(x)k}\right)\right)^{\frac{g(x)}{u(x)}}. \tag{14}$$

Since $\phi(x) \leq g(x)/u(x)$, by setting $r(x) = \frac{g(x)}{u(x)}$ and observing that $g(x) \geq g(x+1) = g(x) - f(x) \geq \frac{3}{4}g(x)$ for any stage $S_x \in H$, we get

$$r(x+1) \geq \frac{r(x)}{4}\left(\max\left(2, \frac{g(x)k}{8\lambda}\right)\right)^{r(x)}. \tag{15}$$

Since $r(1) \geq 3$, we have $r(2) \geq 4$. Therefore, for $x \geq 2$

$$r(x+1) \geq \left(\max\left(2, \frac{g(x)k}{8\lambda}\right)\right)^{r(x)}. \tag{16}$$

Observe that for all $i$, $g(i) \geq g(i+1)$ (equality holds in the case of transient faults). If $k'$ is the number of *lucky* stages, then by induction we get

$$r(k) \geq \text{tower}\left(\max\left(2, \frac{g(k)k}{32\lambda}\right), k'-2, r(2)\right).$$

Notice that $r(2) \geq 1$, $r(k) \leq m$, and $k \geq k'$. Notice that the required bound follows if $k \leq 16$. Now consider the case where $k > 16$. Therefore, we get

$$m \geq \text{tower}\left(\max\left(2, \frac{g(k)k}{8\lambda}\right), k'-1, 1\right) \geq \text{tower}\left(2, k'-2, \max\left(2, \frac{g(k)}{\lambda}\right)\right).$$

Taking the $\log^*$ of both sides and applying Lemma 2.9, we get

$$\log^* m \geq k' + \log^*\left(\max\left(2, \frac{g(k)}{\lambda}\right)\right) + O(1).$$

Suppose $\lambda \leq m/3$; then $g(k) \geq 2m/3$ and so $\frac{g(k)}{\lambda} \geq 2$. On the other hand, if $\lambda > m/3$, we have that $\max(2, \frac{g(k)}{\lambda})$ is in the range $[2, 3]$. Hence, the expected number of stages in $H$ is $O(\log^* m - \log^*(\frac{m}{\lambda}))$. □

THEOREM 2.35. *The competitive ratio of* ROTARY *is* $O(\max(\log^* m - \log^* \frac{m}{\lambda}, \frac{m}{\eta}))$.

*Proof.* The result follows from Lemmas 2.29 and 2.34. □

## 3. Average completion time.

We present a nonclairvoyant deterministic algorithm that is $O(\frac{m}{\eta})$-competitive with respect to average completion time. We then show that every randomized algorithm for this problem has a competitive ratio, with respect to average completion time, that is, $\Omega(R\frac{m}{\eta})$. We assume static release times and $m$ identical processors and consider only permanent faults. We also assume that we have a lower bound of 1 on the length of any task. This is not an unreasonable restriction since we could take our unit of length to be the time to execute one instruction on the fastest processor.

**3.1. Online algorithm.** Geometric Rotary. The algorithm consists of stages. Stage $S_0$ starts when all the jobs are released. Let $u(i)$ denote the number of jobs that have not yet been run to completion at the start of stage $S_i$. We call $S_i$ *full* if $u(i) \geq 2m$.

We now define the stage $S_i$. We first form a queue $Q$ of unfinished jobs. For a full stage, we add exactly one copy of each of the $u(i)$ jobs to the queue. In the case of a stage that is not full, we add $\lceil 2m/u(i) \rceil$ copies of each of these $u(i)$ jobs to $Q$.

During the stage when a processor $P$ becomes idle, the next job $J$ on $Q$ is removed from $Q$ and run from its beginning on $P$ until one of the following three conditions holds:

(a) job $J$ is completed, or
(b) $P$ ran $J$ for $2^i$ time units without completion, or
(c) a fault occurs while $P$ is processing $J$.

If case (a) occurs, $P$ immediately attempts to get another job from $Q$. If case (b) occurs, $P$ suspends $J$ and then immediately attempts to get another job from $Q$. If $Q$ is empty, $P$ idles until the end of the stage. Stage $S_i$ ends when all processors are idle and $Q$ is empty.

**3.2. Analysis.**

Fact 3.1. *Suppose that* $n$ *jobs of length* $\ell_1, \ell_2, \ldots, \ell_n$ *arrive at time* 0. *Let* $\ell_i \leq \ell_j$ *for* $1 \leq i \leq j \leq n$. *The optimal offline average completion time on a single unit speed processor is* $\frac{1}{n}\sum_{i=1}^{n}(n-i+1)\ell_i$.

First notice that doubling the size of jobs only doubles the average offline optimal completion time. This also holds for the multiprocessor case since the shortest processing time algorithm is optimal. So, for the purpose of our analysis, let us assume that we round the execution time of each job up to the next highest integer power of 2. Then for every job, if the length of the job is $\ell = 2^i$, we assume that its length is $2\ell$ (which may be up to 4 times its original length) when we calculate the optimal total completion time. We do so for the ease of analysis, and it is important to observe that the competitive ratio will be off by a factor of at most 2. This doubling idea has been used before; see, for example, [10].

Observe that Geometric Rotary "estimates" the length of a job by running it for a duration of at most $2^i$ during the $i$th stage. Based on this estimated length, we calculate a lower bound on the completion time for the offline optimal algorithm. As we move from one stage to another, the estimated lower bound increases due to an increase in the estimated length of some jobs from $2^i$ to $2^{i+1}$.

We use the terminology "increase in completion time experienced by Geometric Rotary during $S_i$" to mean the sum over all jobs $J$ of the portion of the competition time for $J$ that occurs during stage $S_i$.

At the beginning of a stage $S_i$, the estimated length of an unfinished job is $2^i$. Assuming that this estimated length is correct, one can compute the optimal total completion time. At the end of the stage $S_i$, Geometric Rotary will increase the estimated length of some unfinished jobs to $2^{i+1}$. As a consequence of this increase in estimated lengths of some jobs, there will be an increase in our lower bound of the optimal total competition time. This is what we call the "increase in the estimated lower bound on the completion time of the offline algorithm during a stage."

We will now compare the increase in completion time experienced by Geometric Rotary during $S_i$ to the corresponding increase in the estimated lower bound on the completion time of the offline optimal algorithm during this stage.

LEMMA 3.2. *If we have $k$ jobs of length $x$ to be run on $p$ processors, then every schedule (that doesn't unnecessarily idle the processors) is optimal and has total completion time*

$$\sum_{i=1}^{\lfloor k/p \rfloor} ipx + (k \bmod p)\lceil k/p \rceil x = \lfloor k/p \rfloor(\lfloor k/p \rfloor + 1)px/2 + (k \bmod p)\lceil k/p \rceil x.$$

*Proof.* The $i$th term in the sum represents the sum of completion times of the $i$th jobs executed on each of the $p$ processors. ∎

LEMMA 3.3. *If $S_i$ is full, then* GEOMETRIC ROTARY *guarantees that the increase in the total completion time experienced during $S_i$ is at most $2^{i+1}u(i)^2/\eta$.*

*Proof.* Since processors work in parallel, more processors imply less increase in completion time. At least $\eta$ processors work without fault during this stage. The increase in the completion time is maximized if we have only $\eta$ working processors. Thus the length of $S_i$ is at most $2^i\lceil u(i)/\eta \rceil \leq 2^{i+1}u(i)/\eta$. If all jobs are delayed this amount, then the increase in total completion time is $2^{i+1}u(i)^2/\eta$. ∎

LEMMA 3.4. *If $S_i$ is not full, then* GEOMETRIC ROTARY *guarantees that the increase in total completion time experienced during $S_i$ is at most $5u(i)m2^i/\eta$.*

*Proof.* The length of $S_i$ is at most $2^i\lceil u(i)\lceil 2m/u(i) \rceil/\eta \rceil \leq 2^i\lceil 4m/\eta \rceil \leq 5m2^i/\eta$. Since at most $u(i)$ jobs are delayed by this amount, we get the desired bound. ∎

LEMMA 3.5. *Assume we double the length of $k$ jobs from $x$ to $2x$.*

(a) *If $k \geq m$, then the increase in the offline optimal total completion time is at least $k^2x/8m$.*

(b) *If $k < m$, then the increase in the offline optimal total completion time is at least $kx$.*

*Proof.* Consider only the time required to run the additional $x$ units of each job. By Lemma 3.2 this is $\lfloor k/m \rfloor(\lfloor k/m \rfloor + 1)mx/2 + (k \bmod m)\lceil k/m \rceil x$. Suppose $k \geq m$; substituting $\lfloor k/m \rfloor \geq k/2m$ we get the desired result. If $k < m$, we get the desired bound since $(k \bmod m)\lceil k/m \rceil x = kx$. ∎

LEMMA 3.6. *In every stage $S_i$,* GEOMETRIC ROTARY *guarantees that at least half of the $u(i)$ jobs have at least one copy run the full $2^i$ time units on some processor.*

*Proof.* Recall our assumption that the length of a job is a power of 2. If $S_i$ is full, then this follows because there are at most $m$ faults. If $S_i$ is not full, then the number of jobs that can have all copies experience faults is at most $m/\lceil 2m/u(i) \rceil \leq u(i)/2$. ∎

THEOREM 3.7. *Given static release times,* GEOMETRIC ROTARY *is $O(\frac{m}{\eta})$-competitive, with respect to average completion time, for $\lambda = m - \eta$ permanent faults.*

*Proof.* For stage $S_i$, Lemma 3.6 shows that GEOMETRIC ROTARY runs at least half the jobs for their full $2^i$ time slice. Also, recall our assumption that whenever GEOMETRIC ROTARY runs and completes a job $J$ of length $2^i$, the length of $J$ for the adversary is $2^{i+1}$, and so $J$ can be considered as a job whose length increases from $2^i$ to $2^{i+1}$ when we apply Lemma 3.5. As a consequence, when we apply Lemma 3.5, we can set $k \geq u(i)/2$.

Suppose $u(i) \geq 2m$. Lemmas 3.3 and 3.5 show that the ratio of the increase in the completion time experienced by GEOMETRIC ROTARY and the optimal offline algorithm is at most $\frac{2^{i+1}u(i)^2/\eta}{k^2 2^i/8m}$, where $k \geq u(i)/2 \geq m$. Therefore, the ratio is at most $64m/\eta$.

Suppose $m \leq u(i) < 2m$. Lemmas 3.4 and 3.5 show that the ratio of the increase in the completion time experienced by GEOMETRIC ROTARY and the optimal offline algorithm is at most $\frac{5u(i)m2^i/\eta}{k^2 2^i/8m} \leq \frac{160m^2}{u(i)\eta} \leq 160m/\eta$.

Finally, suppose $u(i) < m$. Lemmas 3.4 and 3.5 show that the ratio of the increase in the completion time experienced by Geometric Rotary and the optimal offline algorithm is at most $10m/\eta$. ∎

Theorem 3.8. *With static release times and related processors, every randomized algorithm is $\Omega(R\frac{m}{\eta})$-competitive with respect to average completion time.*

*Proof.* Using Yao's technique [12], it suffices to bound the expected competitive ratio of any deterministic algorithm where the occurrence of fault is selected from some fixed probability distribution.

First we assume that $\lambda \leq m/2$. The desired lower bound is $\Omega(R)$. There are three processors with speed $R$ and $m - 3$ processors with speed 1. Initially, two jobs of length $R$ arrive. Just before the end of the first unit length stage, the adversary faults one of the three speed $R$ processors uniformly at random. As a consequence, with probability 1/3 there is an unfinished job at the end of the first stage. Now, at the end of the first stage, the remaining two speed $R$ processors fault. One can see that $OPT = 1$, and that the expected completion time of any online algorithm is $\Omega(R)$.

We now assume that $\lambda > m/2$. There are $\lambda$ processors with speed $R$ and $m - \lambda$ with speed 1. There are exactly $\lambda/2$ jobs of length $R$. Just before the end of the first stage, the adversary faults each of the speed $R$ processor independently with probability 1/3. Applying Lemma 2.13 we get the following:

- With probability $1 - O(1/\lambda)$, the number of faults does not exceed $\lambda/2$ before the end of the first stage.
- With probability $1 - O(1/\lambda)$, the number of jobs that cannot be finished before the end of the second stage is $\Omega(\lambda)$.

Since $\lambda > m/2$, with probability $1 - o(1)$, the number of unfinished jobs at the beginning of the second stage is $\Theta(\lambda)$ and at least $\lambda/2$ speed $R$ processors ran without fault for one full stage (and so $OPT = 1$).

At the end of the first stage, the adversary faults the remaining speed $R$ processors. Now there are $\eta = m - \lambda$ unit speed processors, $x = \Theta(\lambda)$ unfinished jobs of length $R$. Therefore, the total completion time for any deterministic online algorithm is at least

$$\sum_{i=1}^{\lfloor x/\eta \rfloor} Ri\eta = \Theta\left(\frac{R\lambda^2}{\eta}\right)$$

with probability $1 - o(1)$. Note that for the case under consideration, the number of jobs is $\Theta(\lambda)$ and $\lambda = \Theta(m)$. Therefore average completion time is $\Omega(R\frac{m}{\eta})$. The result follows since the competitive ratio (against oblivious adversary) is at least $E[\frac{\text{makespan of } \mathcal{A}}{\text{makespan of } OPT}]$. ∎

**4. Transient faults.** In this section we analyze the effect of transient faults, which by definition have duration 0. Here $\lambda$ denotes the total number of faults. Note that it is possible for the number of faults $\lambda$ to exceed $m$. We assume identical processors. The lower bound for makespan with transient faults was handled in section 2.1, so we begin with the upper bound for makespan.

We modify Rotary to immediately end a stage $S$; i.e., every processor suspends its current job if more than $m/2$ faults occur during this stage. We call the new algorithm Transient Rotary. Analogously, we also modify Deterministic Rotary and call this new algorithm Deterministic Transient Rotary. Because of the great similarity with the case of permanent faults, we will be brief in our discussion. Recall Definitions 2.19 and 2.23. Note that $g(i) = m$ for all $i$.

LEMMA 4.1. TRANSIENT ROTARY *guarantees that the length of any stage is at most* $O(OPT)$.

*Proof.* The proof is analogous to that of Lemma 2.21. It is important to observe that all $m$ (instead of at least $\eta$) processors are available at the start of the stage. Also at least $m/2$ (instead of $\eta$) processors are working throughout a stage. ☐

LEMMA 4.2. TRANSIENT ROTARY *guarantees that the total number of stages* $S_i$ *where* $f(i) \geq \frac{m}{4}$ *is at most* $\frac{4\lambda}{m} = O(\frac{\lambda}{m})$.

*Proof.* If more than $\frac{4\lambda}{m}$ stages have $\frac{m}{4}$ or more faults each, then the total number of faults will exceed $\lambda$. ☐

LEMMA 4.3. TRANSIENT ROTARY *guarantees that there will be at most* $m/4$ *jobs left unfinished after* $O(\frac{\lambda}{m})$ *stages starting from the arrival of the last job.*

*Proof.* After the arrival of the last job, consider a stage with total number of faults less than $m/4$. Clearly, there are at most $m/4$ unfinished jobs after this stage. Applying Lemmas 4.1 and 4.2, we get the desired result. ☐

THEOREM 4.4. *The competitive ratio, with respect to makespan, of* DETERMINISTIC TRANSIENT ROTARY *is* $O(\max(\frac{\log m}{\log(\frac{m \log m}{\lambda})}, \frac{\lambda}{m}))$ *for* $\lambda$ *transient faults.*

*Proof.* Given the previous three lemmas, it suffices to bound $|H|$. We ignore the progress made by the online algorithm on stages that are not in $H$. Let $H = \{S_1, S_2, \ldots, S_k\}$. For $1 \leq i \leq k$, the algorithm guarantees that $\phi(i) \geq \lfloor \frac{m}{u(i)} \rfloor$ and $u(i+1) \leq f(i)/\phi(i)$. Therefore, $u(i+1) \leq 2u(i)f(i)/m$. We say that a stage in $H$ is *good* if $f(i) \leq 2\lambda/k$. By Lemma 2.1 at least $|H|/2$ stages are good. Suppose $8\lambda/k \geq m$; then $k \leq 8\lambda/m$ and so the result follows. From now on consider the case $2\lambda/k < m/4$. So for a good stage $S_i$ we have $u(i+1) \leq \frac{4\lambda}{km} u(i)$. Proceeding as in the proof of Lemma 2.32, we are left solving the inequality $m \geq k \log \frac{km}{4\lambda}$ for $k$, which yields the fact that $k = O(\frac{\log m}{\log(\frac{m \log m}{\lambda})})$. ☐

THEOREM 4.5. *The competitive ratio, with respect to makespan, of* TRANSIENT ROTARY *is* $O(\max(\log^* m - \log^* \frac{m}{\lambda}, \frac{\lambda}{m}))$ *for* $\lambda$ *transient faults.*

*Proof.* Given the three previous lemmas, it suffices to show that $|H| = O(\max(\log^* m - \log^* \frac{m}{\lambda}, \frac{\lambda}{m}))$. Observe the fact that all $m$ processors are available at the start of each stage. That is, for all $i$, $g(i) = m$ in the proof of Lemma 2.34. The result follows from Lemma 2.34. ☐

As in the case of ROTARY, we modify GEOMETRIC ROTARY to repeat a stage if it experiences more than $m/2$ faults during that stage. Call the new algorithm TRANSIENT GEOMETRIC ROTARY.

THEOREM 4.6. *For static release times, the competitive ratio, with respect to average completion time, of* GEOMETRIC TRANSIENT ROTARY *is* $O(\max(1, \lambda/m))$ *for* $\lambda$ *transient faults.*

*Proof.* Notice that the increase in the completion time experienced by GEOMETRIC TRANSIENT ROTARY during a stage is at most the total completion time of the optimal offline algorithm. There can be at most $2\lambda/m$ stages that experience more than $m/2$ faults. For a stage that experiences fewer than $m/2$ faults, we can apply the analysis found in the permanent fault case where $\eta \geq m/2$. The result then follows. ☐

**5. No redundancy.** In this section we look at the effect of not allowing more than one copy of a job to be run at any one time. We consider only the case of identical processors. We first show that deterministic algorithms can have very high competitive ratios.

THEOREM 5.1. *If no redundancy is allowed, then the competitive ratio with respect to makespan and with respect to average completion time is $\Omega(\lambda)$ for $\lambda$ permanent faults or $\lambda$ transient faults in a system of identical processors.*

*Proof.* Assume that we have one job $J$. Allow a processor to run $J$ and fault it just before it is ready to complete the job. This process can be repeated $\lambda$ times. ∎

We say that an online algorithm $\mathcal{A}$ is *stage-based* if

1. the time spent by $\mathcal{A}$ is partitioned into stages;
2. at the start of each stage, all of the working processors are idle;
3. jobs considered for execution during a stage are exactly those that are released before the start of the stage;
4. it is possible to determine, at the start of the stage, the number of copies of each job that will be run during the stage;
5. the multiplicities of any two jobs during a stage differ by at most 1.

LEMMA 5.2. ROTARY, TRANSIENT ROTARY, GEOMETRIC ROTARY, *and* GEOMETRIC TRANSIENT ROTARY *are stage-based algorithms.*

For any *stage-based* deterministic algorithm $\mathcal{A}$, we define a randomized algorithm ONE-COPY-$\mathcal{A}$ that does not run multiple copies of a job simultaneously. At the start of a stage $S$, assume that $\mathcal{A}$ plans to run $k$ copies of a job $J_i$ during $S$. ONE-COPY-$\mathcal{A}$ selects an integer uniformly at random from the range 1 to $k$ inclusive, and assigns it to a variable $\alpha_i$. ONE-COPY-$\mathcal{A}$ then simulates $\mathcal{A}$ and when $\mathcal{A}$ makes the $\alpha_i$th assignment of $J_i$, ONE-COPY-$\mathcal{A}$ makes its only assignment of $J_i$ during this stage. The following lemma is the key to analyzing ONE-COPY-$\mathcal{A}$.

LEMMA 5.3. *Let $\mathcal{A}$ be a deterministic* stage-based *online algorithm that starts a stage $S_i$ with $u(i)$ jobs. Assume that for any pattern of $f(i)$ faults during this stage, $\mathcal{A}$ guarantees that at most $u'(i)$ jobs are unfinished at the end of the stage. Then the expected number of unfinished jobs by* ONE-COPY-$\mathcal{A}$ *is $O(u'(i))$ and the variance of the number of unfinished jobs is not greater than the expected value.*

*Proof.* Let $\mathcal{J} = \{J_1, J_2, \ldots, J_{u(i)}\}$ be the set of unfinished jobs at the start of the stage $S_i$. Let $\phi(J_j)$ be the number of copies of a job $J_j \in \mathcal{J}$ run by $\mathcal{A}$ in this stage. Without loss of generality, let $0 < \phi(J_a) \leq \phi(J_b)$ for $1 \leq a \leq b \leq u$. Let $x_a = 1$ if $J_a$ is not finished by ONE-COPY-$\mathcal{A}$ during the stage, and let $x_a = 0$ otherwise. Let $x = \sum_{i=1}^{u} x_a$ denote the number of jobs that are not completed during the stage by ONE-COPY-$\mathcal{A}$. Let $f_j(i)$ be the number of processors experiencing a fault while running a copy of $J_j$ using algorithm $\mathcal{A}$. First,

$$E[x] = \sum_{j=1}^{u(i)} E[x_j] = \sum_{j=1}^{u(i)} \frac{f_j(i)}{\phi(J_j)} \leq \sum_{j=1}^{u(i)} \frac{f_j(i)}{\phi(J_1)}.$$

Note that since in the worst case the adversary for $\mathcal{A}$ will fault all the low multiplicity jobs,

$$\sum_{j=1}^{u'(i)+1} \phi(J_j) > f(i) = \sum_{j=1}^{u(i)} f_j(i).$$

Since the multiplicity of each job is at most $\phi(J_1) + 1$, we get that

$$E[x] \leq \frac{1}{\phi(J_1)} \sum_{j=1}^{u(i)} f_j(i) \leq \frac{1}{\phi(J_1)} \sum_{j=1}^{u'(i)+1} \phi(J_j) \leq \frac{1+\phi(J_1)}{\phi(J_1)}(u'(i)+1) \leq 4u'(i).$$

For fixed $f_j(i)$'s, the $x_j$'s are independent. As a consequence, the variance of $x$ (the sum of 0/1-random variables) does not exceed the expected value of $x$. $\square$

THEOREM 5.4. *The randomized competitive ratio, with respect to makespan, of the algorithm* ONE-COPY-DETERMINISTIC ROTARY *is* $O(\max(\frac{\log m}{\log(\frac{m \log m}{\lambda})}, \frac{m}{\eta}))$ *for permanent faults and* $O(\max(\frac{\log m}{\log(\frac{m \log m}{\lambda})}, \frac{\lambda}{m}))$ *for transient faults in a system of identical processors.*

*Proof.* The analysis is analogous to the analysis of ROTARY. The major difference is in applying the proof of Lemma 2.34, which establishes a bound on $H$.

Of course, the bound we wish to establish is different from the one proved in Lemma 2.34. The recurrence relation established in (14) will be different in the current proof. We continue to follow the notation established in the proofs of Lemmas 5.3 and 2.34. Notice that $E(x) \leq \frac{1}{\phi(J_1)} f(i)$. Let $g(i)$ be the number of processors that did not experience a fault during the $i$th stage (i.e., $S_i$). Notice that $g(i)/u(i) \geq \phi(J_1) \geq \lfloor g(i)/u(i) \rfloor \geq 3$. As a consequence, $E(x) \leq \epsilon u(i) \frac{f(i)}{g(i)}$, where $\epsilon < 3/2$.

Recall that a stage $S_i \in H$ is defined to be *good* (in the proof of Lemma 2.34) if $f(i) \leq 2\alpha/k \leq 2\lambda/k$. We define a stage to be *unlucky* if $u(i+1) > 2E[u(i+1)]$. Therefore, at the end of a *lucky* stage $u(i+1) \leq 2\epsilon u(i) \frac{f(i)}{g(i)}$. Therefore, $u(i+1) \leq 2\epsilon u(i) \left(\min(\frac{1}{4}, \frac{4\lambda}{kg(i)})\right)$.

As before, at least $k/2$ stages in $H$ are *lucky*. Consider the rate at which the $u(i)$'s decrease on *lucky* stages. Applying the fact that $u(1) \leq m$ and $u(k) \geq 1$, we get $1 \leq u(k) \leq u(1)[2\epsilon \min(\frac{1}{4}, \frac{4\lambda}{kg(k)})]^{k/2}$. Solving for $k$ we get $k = \frac{\log m}{\log(\frac{m \log m}{\lambda})}$. $\square$

THEOREM 5.5. *If no redundancy is allowed, then the optimal randomized competitive ratio, with respect to makespan, is* $\Omega(\max(\frac{\log m}{\log(\frac{m \log m}{\lambda})}, \frac{m}{\eta}))$ *for permanent faults and* $\Omega(\max(\frac{\log m}{\log(\frac{m \log m}{\lambda})}, \frac{\lambda}{m}))$ *for transient faults in a system of identical processors.*

*Proof.* We convert DETERMINISTIC-DETER$(m, m, \lambda, 1, \mathcal{A})$ into a randomized adversary by selecting the faults in each stage uniformly at random among all working processors.

Suppose $\lambda \leq m - \frac{m \log \log m}{\log m}$. Therefore, case 1 of the adversary applies. Let $u(i)$ be a random variable that represents the number of unfinished jobs at the start of the $i$th stage. Since $\lambda/\log m$ processors are chosen uniformly at random, the probability that a given processor experiences a fault during this stage is at least $\frac{\lambda}{m \log m}$. Therefore $E[u(i+1)] \geq u(i) \frac{\lambda}{m \log m}$ and $\text{VAR}[u(i+1)] \leq E[u(i+1)]$.

Let $k$ be an integer such that $E[u(k)] > \sqrt{m}$ and $E[u(k+1)] \leq \sqrt{m}$. We now argue that with probability $1 - o(1)$, $k = \Omega(\frac{\log m}{\log(\frac{m \log m}{\lambda})})$. As in the proof of Lemma 2.14, we define a stage $i$ to be *good* if $|u(i+1) - E[u(i+1)]| \leq E[u(i+1)]/2$. Applying Chebyshev's inequality, with probability $\sum_{i=1}^{x} \frac{4}{E[(u(i+1)]} \leq \frac{4x}{\sqrt{m}}$, we have $|u(i+1) - E[u(i+1)]| \leq \frac{1}{2} E[u(i+1)]$ for all $i$ in the range 1 through $x$ $(\leq k)$. Substituting $u(1) = m$, we get $u(i+1) \geq \frac{u(i)}{2} \frac{\lambda}{m \log m} \geq m(\frac{\lambda}{2m \log m})^i$. Since $u(k+1) \leq \sqrt{m}/2$, we get $k = \Omega(\frac{\log m}{\log(\frac{m \log m}{\lambda})})$. The result follows since $OPT$ is $O(1)$.

Now, suppose $\lambda > m - \frac{m \log \log m}{\log m}$. Notice that during the first two stages, the probability that a working processor faults is $\Theta(1)$. Therefore with probability $1 - o(1)$, $\Theta(m)$ jobs will not be completed at the end of the second stage. The desired bound for the permanent fault case follows, since there are exactly $\eta$ working processors and $\Theta(m)$ unfinished jobs. On the other hand, in the case of transient faults, no progress is made for the next $\lfloor (\lambda - m)/m \rfloor$ stages since all $m$ processors fault just before the end of each of those stages. The result then follows. Note that the analysis is analogous to that in the proof of Lemma 2.4. □

THEOREM 5.6. *If no redundancy is allowed, then the optimal randomized competitive ratio, with respect to average completion time, is* $\Theta(\max(\frac{m}{\eta}, 1))$ *for permanent faults.*

*Proof.* The lower bound follows from Theorem 3.8. Applying Lemma 5.3, one can use the analysis of GEOMETRIC ROTARY to get the desired bound. □

**Acknowledgments.** We thank Taieb Znati for pointing out to us the need for a theoretical analysis of fault tolerant scheduling. We also thank Joel Wein, Sundar Vishwanathan, Sunondo Ghosh, and Daniel Mossé for helpful discussions. Finally, we would like to thank the referees for many useful comments and suggestions.

## REFERENCES

[1] N. ALON AND J. H. SPENCER, *The Probabilistic Method*, Wiley-Intersci. Ser. Discrete Math. Optim., John Wiley & Sons, New York, 1992.

[2] S. BEN-DAVID, A. BORODIN, R. KARP, G. TARDOS, AND A. WIGDERSON, *On the power of randomization in on-line algorithms*, Algorithmica, 11 (1994), pp. 2–14.

[3] A. BORODIN AND R. EL-YANIV, *On randomization in on-line computation*, in Proceedings of the IEEE Conference on Computational Complexity, 1996, pp. 226–238.

[4] T. CORMEN, C. LEISERSON, AND R. RIVEST, *Introduction to Algorithms*, MIT Press, Cambridge, MA, 1990.

[5] A. FELDMANN, J. SGALL, AND S. TENG, *Dynamic scheduling on parallel machines*, Theoret. Comput. Sci., 130 (1994), pp. 49–72.

[6] R. GRAHAM, E. LAWLER, J. LENSTRA, AND A RINNOOY KAN, *Optimization and approximation in deterministic sequencing and scheduling: A survey*, Ann. Discrete Math., 5 (1979), pp. 287–326.

[7] T. MATSUMOTO, *Competitive analysis of the round robin algorithm*, in Algorithms and Computation (Nagoya, 1992), Lecture Notes in Comput. Sci. 650, Springer-Verlag, Berlin, 1992, pp. 71–77.

[8] R. MOTWANI, S. PHILLIPS, AND E. TORNG, *Non-clairvoyant scheduling*, Theoret. Comput. Sci., 130 (1994), pp. 17–47.

[9] D. SLEATOR AND R. TARJAN, *Amortized efficiency of list update and paging rules*, Comm. ACM, 28 (1985), pp. 202–208.

[10] D. SHMOYS, J. WEIN, AND D. WILLIAMSON, *Scheduling parallel machines on-line*, in Proceedings of the IEEE Symposium on Foundations of Computing, 1991, pp. 131–140.

[11] J. VYTOPIL, ED., *Formal Techniques in Real-Time and Fault-Tolerant Systems*, Kluwer Academic Publishers, Norwell, MA, 1993.

[12] A. YAO, *Probabilistic computations: Towards a unified measure of complexity*, in Proceedings of the IEEE Symposium on Foundations of Computing, 1977, pp. 222–227.

SIAM J. COMPUT.
Vol. 34, No. 3, pp. 720–742

# CLASSIFYING THE COMPLEXITY OF CONSTRAINTS USING FINITE ALGEBRAS[*]

ANDREI BULATOV[†], PETER JEAVONS[‡], AND ANDREI KROKHIN[§]

**Abstract.** Many natural combinatorial problems can be expressed as constraint satisfaction problems. This class of problems is known to be **NP**-complete in general, but certain restrictions on the form of the constraints can ensure tractability. Here we show that any set of relations used to specify the allowed forms of constraints can be associated with a finite universal algebra and we explore how the computational complexity of the corresponding constraint satisfaction problem is connected to the properties of this algebra. Hence, we completely translate the problem of classifying the complexity of restricted constraint satisfaction problems into the language of universal algebra.

We introduce a notion of "tractable algebra," and investigate how the tractability of an algebra relates to the tractability of the smaller algebras which may be derived from it, including its subalgebras and homomorphic images. This allows us to reduce significantly the types of algebras which need to be classified. Using our results we also show that if the decision problem associated with a given collection of constraint types can be solved efficiently, then so can the corresponding search problem. We then classify all finite strictly simple surjective algebras with respect to tractability, obtaining a dichotomy theorem which generalizes Schaefer's dichotomy for the generalized satisfiability problem. Finally, we suggest a possible general algebraic criterion for distinguishing the tractable and intractable cases of the constraint satisfaction problem.

**Key words.** constraint satisfaction problem, universal algebra, dichotomy theorem

**AMS subject classifications.** 08A70, 68Q25, 68T99

**DOI.** 10.1137/S0097539700376676

## 1. Introduction.

In a *constraint satisfaction problem* the aim is to find an assignment of values to a given set of variables, subject to *constraints* on the values which can be assigned simultaneously to certain specified subsets of the variables [39, 43]. One common example of such a problem is the standard propositional satisfiability problem [21], where the variables must be assigned Boolean values and the constraints are specified by clauses.

The mathematical framework used to describe constraint satisfaction problems has strong links with many other areas of computer science and mathematics. For example, links with relational database theory [22, 23, 34], with some notions of logic and group theory [1, 18, 20], and with universal algebra [31] have been investigated. There is an early survey of these results in [46].

Throughout the paper we assume that $\mathbf{P} \neq \mathbf{NP}$, and we call a problem *tractable* only if it belongs to $\mathbf{P}$. The constraint satisfaction problem is known to be **NP**-hard in general [39, 43]. However, certain restrictions on the form of the constraints have been shown to ensure tractability [8, 10, 30, 32, 33, 56].

One fundamental open research problem in this area is to characterize exactly the forms of constraint relations that give rise to tractable problem classes. This problem is important from a theoretical perspective, as it helps to clarify the boundary

[*]Received by the editors August 15, 2000; accepted for publication (in revised form) October 19, 2004; published electronically April 19, 2005. This work was supported by the UK EPSRC, under grants GR/M12926 and GR/R29598.

http://www.siam.org/journals/sicomp/34-3/37667.html

[†]School of Computing Science, Simon Fraser University, Canada (abulatov@cs.sfu.ca).

[‡]Computing Laboratory, University of Oxford, Oxford, UK (Peter.Jeavons@comlab.ox.ac.uk).

[§]Department of Computer Science, University of Durham, Durham, UK (Andrei.Krokhin@durham.ac.uk).

between tractability and intractability in a wide range of combinatorial search problems. It is also important from a practical perspective, as it allows the development of constraint programming languages that exploit the existence of diverse families of tractable constraints to provide more efficient solution techniques [38, 50].

The problem of characterizing the tractable cases was completely solved for the important special case of Boolean constraint satisfaction problems by Schaefer in 1978 [52]. Schaefer established that for Boolean constraint satisfaction problems (which he called "generalized satisfiability problems"), there are exactly six different families of tractable constraints, and any problem involving constraints not contained in one of these six families is **NP**-complete. This important result is known as Schaefer's dichotomy theorem. Similar dichotomy results have also been obtained for other combinatorial problems over a Boolean domain that are related to the Boolean constraint satisfaction problem [11].

Schaefer [52] raised the question of how the analysis of complexity could be extended to larger sets of possible values (that is, sets with more than two elements). Some progress has been made with this question recently, and a number of tractable families of constraints have been identified, over both finite and infinite sets. In particular, Feder and Vardi [20] used techniques from logic programming and group theory to identify three broad families of tractable constraints, which include all of Schaefer's six classes. A series of papers by Jeavons and coauthors has shown that any individual tractable constraint class over a finite domain can be characterized using algebraic properties of relations [28, 29, 30, 32]. This approach was used by Bulatov to obtain a complete classification for the complexity of constraints on a three-element set [3]. Finally, we note that for temporal and spatial reasoning problems involving constraints over infinite sets of values, several tractable constraint classes have been identified [17, 44, 51], and a dichotomy theorem has been obtained for qualitative temporal reasoning problems over intervals expressed using Allen's interval algebra [35]. It has also been shown recently that the complexity of certain forms of constraint over infinite sets of values can be analyzed using algebraic properties [2].

However, there is still no complete classification for the complexity of constraints over finite sets with more than three elements, and no dichotomy has so far been established for arbitrary finite sets.

In our opinion, the main difficulty in addressing this question is the lack of a powerful and convenient language in which the properties of constraint satisfaction problems responsible for complexity can be expressed. Schaefer's dichotomy theorem is stated in terms of the syntactic properties of propositional forms, which is certainly not an appropriate language for non-Boolean constraints. A number of different attempts have been made to find such an appropriate language [13, 14, 20, 28, 30]; we believe that the most fruitful of these is the one that uses the algebraic properties of constraints [28, 30].

The first step in the algebraic approach exploits the well-known idea that, given an initial set of constraint relations, there will often be further relations that can be added to the set without changing the complexity of the associated problem class. In fact, it has been shown that it is possible to add all the relations that can be derived from the initial relations using certain simple rules. The larger sets of relations obtained using these rules are known as *relational clones* [16, 48]. Hence the first step in the analysis is to note that it is sufficient to analyze the complexity only for those sets of relations which are relational clones.

The next step in the algebraic approach is to note that relational clones can be characterized by their *polymorphisms*, which are algebraic *operations* on the same

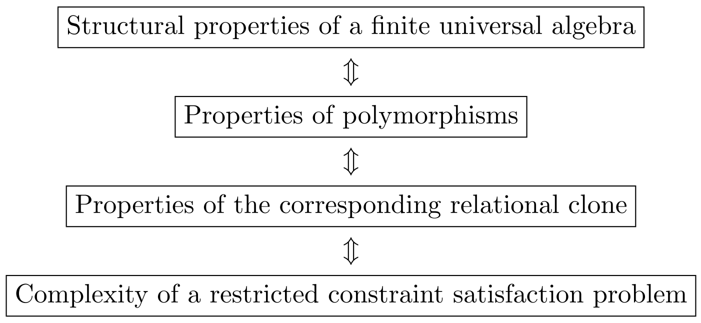

Structural properties of a finite universal algebra
Properties of polymorphisms
Properties of the corresponding relational clone
Complexity of a restricted constraint satisfaction problem

FIG. 1.1. *Translating questions about the complexity of different forms of constraints into questions about the properties of algebras.*

underlying set [27, 30]. As well as providing a convenient and concise method for describing large families of relations, the polymorphisms also reflect certain aspects of the structure of the relations that can be used for designing efficient algorithms. This link between relational clones and polymorphisms has already played a key role in identifying many tractable constraint classes and developing appropriate efficient solution algorithms for them [3, 4, 6, 7, 12, 28].

However, as we shall see later on in this paper, working directly with the polymorphisms of a set of relations is sometimes not the most convenient and powerful way to investigate the complexity of the corresponding constraint satisfaction problems. In this paper we take the algebraic approach one step further by linking constraint satisfaction problems with finite universal algebras (see Figure 1.1). We show that the language of finite algebras provides a number of very powerful new tools for analyzing the complexity of constraint problems. In particular, using this language allows us to suggest a simple criterion for distinguishing tractable and intractable cases. This criterion makes extensive use of notions related to universal algebras, and is difficult to formulate concisely without using this language. Another advantage of this new translation is that it allows us to make use of the deep structural results developed for classifying the structure of finite algebras [25, 42, 53].

As the first fruit of using this machinery we exhibit a new dichotomy theorem for a class of algebras which properly includes all the two-element algebras, and hence provide a true generalization of Schaefer's dichotomy theorem.

The paper is organized as follows. In section 2 we define the class of constraint satisfaction problems we are considering, where the constraints are chosen from a specified set of relations. Then we define the way in which these sets of relations can be classified according to the complexity of the corresponding constraint satisfaction problems. We show how this question can be translated into an equivalent question about relational clones, and hence further translated into a question about classifying sets of operations. In section 3 we make the new step from sets of operations to finite algebras, which is the real focus of this paper. Using this final translation we introduce the notion of a tractable algebra. We are then able to restate Schaefer's dichotomy theorem in a much shorter and possibly clearer form, as a classification of two-element algebras with respect to tractability. In section 4 we prove that, in this context, it suffices to consider certain restricted classes of algebras. As a by-product we show that, if the decision problem for a set of constraint types can be solved efficiently, then so can the corresponding search problem. In section 5 we study how the tractability of a finite algebra relates to the tractability of its smaller derived algebras. In section 6 we use the results obtained to classify all strictly simple surjective algebras. Finally, in section 7, we use the new algebraic terminology to state a conjecture about the general structure of tractable algebras, and provide examples to illustrate and support this conjecture.

## 2. Definitions and earlier results.

### 2.1. Constraint satisfaction problems.

The central notion in the study of constraints and constraint satisfaction problems is the notion of a *relation*.

DEFINITION 2.1. *For any set $A$, and any natural number $n$, the set of all $n$-tuples of elements of $A$ is denoted by $A^n$. Any subset of $A^n$ is called an $n$-ary* relation *over $A$. The set of all finitary relations over $A$ is denoted by $R_A$. A* constraint language *over $A$ is a subset of $R_A$.*

The "constraint satisfaction problem" was introduced by Montanari [43] in 1974 and has been widely studied [15, 20, 36, 39, 40, 41]. In this paper we study a parameterized version of the standard constraint satisfaction problem, in which the parameter is a constraint language specifying the possible forms of the constraints.

DEFINITION 2.2. *For any set $A$ and any constraint language $\Gamma$ over $A$, the* constraint satisfaction problem $\mathbf{CSP}(\Gamma)$ *is the combinatorial decision problem with*

**Instance:** *A triple $(V, A, \mathcal{C})$, where*

- *$V$ is a set of* variables*;*
- *$\mathcal{C}$ is a set of* constraints*, $\{C_1, \dots, C_q\}$.*
  *Each constraint $C_i \in \mathcal{C}$ is a pair $\langle s_i, \rho_i \rangle$, where*
  - *$s_i$ is a tuple of variables of length $m_i$, called the* constraint scope*;*
  - *$\rho_i \in \Gamma$ is an $m_i$-ary relation over $A$, called the* constraint relation.

**Question:** *Does there exist a* solution*, that is, a function $\varphi$, from $V$ to $A$, such that, for each constraint $\langle s_i, \rho_i \rangle \in \mathcal{C}$, with $s_i = (x_{i_1}, \dots, x_{i_m})$, the tuple $(\varphi(x_{i_1}), \dots, \varphi(x_{i_m}))$ belongs to $\rho_i$?*

In this paper we shall consider only cases where the set $A$, specifying the possible values for the variables, is finite. In such cases the size of a problem instance can be taken to be the length of a string containing all constraint scopes and all tuples of all constraint relations from the instance.

*Example* 2.3. An instance of the standard propositional 3-SATISFIABILITY problem [21, 45] is specified by giving a formula in propositional logic consisting of a conjunction of clauses, each of which contains at most three literals, and asking whether there are values for the variables that make the formula true.

Suppose that $\Phi = F_1 \wedge \cdots \wedge F_n$ is such a formula, where the $F_i$ are clauses. The satisfiability question for $\Phi$ can be expressed as the constraint satisfaction problem instance $(V, \{0,1\}, \mathcal{C})$, where $V$ is the set of all variables appearing in the clauses $F_i$, the values 0 and 1 represent the logical values FALSE and TRUE, and $\mathcal{C}$ is the set of constraints $\{\langle s_1, \rho_1 \rangle, \dots, \langle s_n, \rho_n \rangle\}$, where each constraint $\langle s_k, \rho_k \rangle$ is constructed as follows:

- $s_k = (x_1^k, x_2^k, x_3^k)$, where $x_1^k, x_2^k, x_3^k$ are the variables appearing in clause $F_k$;
- $\rho_k = \{0,1\}^3 \setminus \{(a_1, a_2, a_3)\}$, where $a_i = 1$ if $x_i^k$ is negated in $F_k$ and $a_i = 0$ otherwise (i.e., $\rho_k$ contains exactly those 3-tuples that make $F_k$ true).

The solutions of this instance are exactly the assignments which make the formula $\Phi$ true.

If we define $\Gamma_{\text{3-SAT}}$ to be the constraint language over $\{0,1\}$ consisting of all relations expressible by 3-clauses, then any instance of 3-SATISFIABILITY can be expressed as an instance of $\mathbf{CSP}(\Gamma_{\text{3-SAT}})$ in this way, and vice versa. □

*Example* 2.4. An instance of GRAPH UNREACHABILITY consists of a graph $G = (V, E)$ and a pair of vertices, $v, w \in V$. The question is whether there is no path in $G$ from $v$ to $w$.

This can be expressed as the constraint satisfaction problem instance defined by $(V, \{0, 1\}, \mathcal{C})$, where

$$\mathcal{C} = \{\langle e, \{=_{\{0,1\}}\}\rangle \mid e \in E\} \cup \{\langle (v), \{(0)\}\rangle, \langle (w), \{(1)\}\rangle\},$$

where $=_{\{0,1\}}$ denotes the equality relation over the set $\{0, 1\}$.

If we define $\Gamma_{\text{GU}}$ to be the constraint language over $\{0, 1\}$ containing just the relations $=_{\{0,1\}}$, $\{(0)\}$ and $\{(1)\}$, then any instance of GRAPH UNREACHABILITY can be expressed as an instance of $\mathbf{CSP}(\Gamma_{\text{GU}})$ in this way. $\square$

*Example* 2.5. An instance of GRAPH $q$-COLORABILITY consists of a graph $G$. The question is whether the vertices of $G$ can be labelled with $q$ colors so that adjacent vertices are assigned different colors [21, 45].

This problem corresponds to the problem $\mathbf{CSP}(\{\neq_A\})$, where $A$ is a $q$-element set (of colors) and $\neq_A$ is the disequality relation over $A$, defined by

$$\neq_A \; = \{(a, b) \in A^2 \mid a \neq b\}. \qquad \square$$

*Example* 2.6. Given a fixed graph $H$, an instance of $H$-COLORABILITY consists of a graph $G$. The question is whether there is a mapping from the vertices of $G$ to the vertices of $H$, such that adjacent vertices in $G$ are mapped to adjacent vertices in $H$ [24]. (In the special case when $H$ is the complete graph on $q$ vertices, $K_q$, the $H$-COLORABILITY problem reduces to the GRAPH $q$-COLORABILITY problem described in Example 2.5.)

This problem corresponds to the problem $\mathbf{CSP}(\{E_H\})$, where $E_H$ is the edge relation of $H$, that is, the binary relation consisting of all adjacent pairs of vertices from $H$. $\square$

Many other examples of standard combinatorial problems expressed as constraint satisfaction problems can be found in [31]. For alternative formulations of the constraint satisfaction problem, such as the problem of deciding whether there is a homomorphism from one relational structure to another, see [20, 31, 34].

**2.2. Tractable constraint languages.** Throughout this paper we shall say that a problem is *tractable* if there exists a deterministic polynomial-time algorithm solving all instances of that problem. We can use Definition 2.2 to classify constraint languages according to the complexity of the corresponding constraint satisfaction problem.

In order to be able to classify infinite, as well as finite, constraint languages, we define the notion of a tractable constraint language in a way that depends on finite subsets only.

DEFINITION 2.7. *For any set $A$, a finite constraint language $\Gamma \subseteq R_A$ is said to be* tractable *if* $\mathbf{CSP}(\Gamma)$ *is tractable.*

*An infinite constraint language $\Gamma \subseteq R_A$ is said to be* tractable *if every finite subset of $\Gamma$ is tractable.*

*A constraint language $\Gamma \subseteq R_A$ is said to be* **NP**-complete *if* $\mathbf{CSP}(\Delta)$ *is* **NP**-*complete for some finite $\Delta \subseteq \Gamma$.*

*Example* 2.8. It is well known (see [21, 45]) that the GRAPH $q$-COLORABILITY problem described in Example 2.5 is tractable when $q \leq 2$ and is **NP**-complete otherwise. Hence, it follows from Example 2.5 that the finite constraint language containing just the single relation $\neq_A$ is tractable when $|A| \leq 2$ and is **NP**-complete otherwise. $\square$

*Example* 2.9. The complexity of the $H$-COLORABILITY problem for *undirected* graphs $H$ was completely characterized in [24], where it was shown that if an undirected graph $H$ is bipartite or has a loop, then the corresponding $H$-COLORABILITY problem is tractable; otherwise it is **NP**-complete.

Using this result, it follows from Example 2.6 that a constraint language $\Gamma = \{E\}$, consisting of a single symmetric binary relation $E$, is tractable if $E$ is the edge relation of a bipartite graph, or a graph with a loop. Otherwise $\Gamma$ is **NP**-complete. $\square$

A finite or infinite constraint language $\Gamma$ with the property that $\mathbf{CSP}(\Gamma)$ is tractable will be called *globally tractable*. Clearly any constraint language that is globally tractable is tractable in the sense of Definition 2.7, but it is not immediately clear whether or not the converse holds for all infinite languages. This is because for a certain infinite constraint language $\Gamma$, it may be the case that for each finite subset $\Delta \subseteq \Gamma$ there exists a polynomial-time algorithm $Alg(\Delta)$ solving $\mathbf{CSP}(\Delta)$, and yet there is no *uniform* polynomial-time algorithm solving $\mathbf{CSP}(\Gamma)$. However, we know of no examples where this is the case and we conjecture that any tractable constraint language is also globally tractable.

*Example* 2.10. Let $A$ be a finite field, and let $\Gamma_{\mathrm{LIN}}$ be the constraint language consisting of all relations over $A$ which consist of all solutions to some system of linear equations over $A$. Any relation from $\Gamma_{\mathrm{LIN}}$, and therefore any instance of $\mathbf{CSP}(\Gamma_{\mathrm{LIN}})$, can be represented by a system of linear equations over $A$. Indeed, if $\rho \in \Gamma_{\mathrm{LIN}}$, then it is the solution space of the system of linear equations obtained by the following procedure:

*Step* 1 Pick an arbitrary element $\overline{a}_0 = (a_{01}, \dots, a_{0n}) \in \rho$, and set $\rho_0 = \{\overline{b} - \overline{a}_0 \mid \overline{b} \in \rho\}$.

*Step* 2 For every member $(a_1, \dots, a_n)$ of $\rho_0$, form the equation $a_1x_1 + \cdots + a_nx_n = 0$, and find a basis, $\rho^{\perp}$, of the solution space of the resulting system of equations.

*Step* 3 For each $(b_1, \dots, b_n) \in \rho^{\perp}$, output the equation $b_1x_1 + \cdots + b_nx_n = b_0$, where $b_0 = b_1a_{01} + \cdots + b_na_{0n}$.

Since any instance of $\mathbf{CSP}(\Gamma_{\mathrm{LIN}})$ may be solved in polynomial time (e.g., by Gaussian elimination), it follows that $\Gamma_{\mathrm{LIN}}$ is a (globally) tractable constraint language. $\square$

A constraint language over the set $A = \{0, 1\}$ is known as a *Boolean* constraint language. The complexity of $\mathbf{CSP}(\Gamma)$ has been investigated [52] for all Boolean constraint languages $\Gamma$, and the following complete classification has been obtained.

THEOREM 2.11 (see Schaefer [52]). *A Boolean constraint language, $\Gamma$, is (globally) tractable if (at least) one of the following six conditions holds:*

1. *Every relation in $\Gamma$ contains a tuple in which all entries are $0$;*
2. *Every relation in $\Gamma$ contains a tuple in which all entries are $1$;*
3. *Every relation in $\Gamma$ is definable by a formula in conjunctive normal form in which each conjunct has at most one negated variable;*
4. *Every relation in $\Gamma$ is definable by a formula in conjunctive normal form in which each conjunct has at most one unnegated variable;*
5. *Every relation in $\Gamma$ is definable by a formula in conjunctive normal form in which each conjunct contains at most two literals;*
6. *Every relation in $\Gamma$ is the set of solutions of a system of linear equations over the finite field* GF(2).

*Otherwise it is* **NP**-*complete.*

In particular, Theorem 2.11 establishes that any Boolean constraint language can be classified as either tractable or **NP**-complete, and hence this result is known as Schaefer's dichotomy theorem.

A similar dichotomy theorem has been obtained for constraint languages over any set with three elements [3], using some of the algebraic methods described below. The classification problem for languages over larger finite sets is still open [20], and is the central topic of this paper.

PROBLEM 2.12 ("tractable relations problem"). *Characterize all tractable constraint languages over finite sets.*

*Example* 2.13. One of the first non-Boolean tractable constraint languages to be characterized was the set $\Gamma_{\text{ZOA}}$ of "0/1/all" relations described in [10]. The set $\Gamma_{\text{ZOA}}$ contains all relations over some fixed set $A$ of the following forms:

(i) all unary relations;
(ii) all binary relations of the form $A_1 \times A_2$ for subsets $A_1, A_2$ of $A$;
(iii) all binary relations of the form $\{(a, \pi(a)) \mid a \in A_1\}$ for some subset $A_1$ of $A$ and some permutation $\pi$ of $A$;
(iv) All binary relations of the form $\{(a,b) \in A_1 \times A_2 \mid a = a_1 \vee b = a_2\}$ for some subsets $A_1, A_2$ of $A$ and some elements $a_1 \in A_1, a_2 \in A_2$.

It was shown in [10] that $\mathbf{CSP}(\Gamma_{\text{ZOA}})$ is tractable, and that for any binary relation $\rho$ over $A$ which is *not* in $\Gamma_{\text{ZOA}}$, $\mathbf{CSP}(\Gamma_{\text{ZOA}} \cup \{\rho\})$ is **NP**-complete. □

**2.3. From arbitrary constraint languages to relational clones.** To describe the tractable Boolean constraint languages, Schaefer used syntactic properties of propositional formulas representing Boolean relations. In the non-Boolean case this method can no longer be used. We therefore need an adequate language in which it is possible to express the properties of constraint languages which are responsible for the complexity of the corresponding constraint satisfaction problems.

A useful first step in tackling this problem is to consider what additional relations can be added to a constraint language without changing the complexity of the corresponding problem class. This technique has been widely used in the analysis of Boolean constraint satisfaction problems [11, 52], and in the analysis of temporal and spatial constraints [17, 44, 51]; it was introduced for the study of constraints over arbitrary finite sets in [27].

To use this technique we first define a method for deriving new relations from given ones. The method we use involves defining the new relations using certain kinds of logical formulas involving the given relations. To define such formulas we use the standard correspondence between relations and predicates: a relation consists of all tuples of values for which the corresponding predicate holds. (We will use the same symbol for a predicate and its corresponding relation, since the meaning will always be clear from the context.)

DEFINITION 2.14 (see [48]). *A constraint language $\Gamma \subseteq R_A$ is called a* relational clone *if it contains every relation (predicate) expressible by a first-order formula involving*

(i) *relations (predicates) from $\Gamma \cup \{=_A\}$ (where $=_A$ is the equality relation on the set $A$);*
(ii) *conjunction; and*
(iii) *existential quantification.*

First-order formulas involving only conjunction and existential quantification are often called *primitive positive* (pp) formulas.

For any constraint language $\Gamma$, there is a unique smallest relational clone containing $\Gamma$, which is denoted $\langle\Gamma\rangle$ and is called the relational clone *generated by* $\Gamma$. The set $\langle\Gamma\rangle$ consists of all relations definable by pp-formulas over the relations in $\Gamma$ together with the equality relation.

*Example* 2.15. Consider the Boolean constraint language $\Gamma = \{R_1, R_2\}$, where $R_1 = \{(0,1),(1,0),(1,1)\}$ and $R_2 = \{(0,0),(0,1),(1,0)\}$.

It is straightforward to check that every binary Boolean relation can be expressed by a pp-formula involving $R_1$ and $R_2$. For example, the relation $R_3 = \{(0,0),(1,0),(1,1)\}$ can be expressed by the formula $R_3 = \exists y R_1(x,y) \wedge R_2(y,z)$. Hence the relational clone generated by $\Gamma$, $\langle\Gamma\rangle$, includes all 16 binary Boolean relations.

In fact it can be shown that $\langle\Gamma\rangle$ consists of precisely those Boolean relations (of any arity) that can be expressed as a conjunction of unary or binary Boolean relations [49, 53]. $\square$

There are a number of different but equivalent definitions of relational clones [16, 48], and a different definition was used in [27] to establish the following theorem. We give a proof here that uses Definition 2.14.

THEOREM 2.16 (see [27]). *For any set of relations* $\Gamma$ *and any finite set* $\Delta \subseteq \langle\Gamma\rangle$, *there is a polynomial time reduction from* $\mathbf{CSP}(\Delta)$ *to* $\mathbf{CSP}(\Gamma)$.

*Proof.* Let $\Delta = \{\varrho_1, \ldots, \varrho_k\}$ be a finite set of relations over the finite set $A$, where each $\varrho_i$ is expressible by a pp-formula involving relations from $\Gamma$ and the equality relation, $=_A$. Note that we may fix these representations.

Any instance $(V; A; \mathcal{C}) \in \mathbf{CSP}(\Delta)$ can be transformed as follows. For every constraint $\langle s, \rho\rangle \in \mathcal{C}$, where $s = (v_1, \ldots, v_l)$ and $\rho$ is representable by the pp-formula

$$\rho(v_1, \ldots, v_l) = \exists u_1, \ldots, u_m \; (\rho_1(w_1^1, \ldots, w_{l_1}^1) \wedge \cdots \wedge \rho_n(w_1^n, \ldots, w_{l_n}^n)),$$

where $w_1^1, \ldots, w_{l_1}^1, \ldots, w_1^n, \ldots, w_{l_n}^n \in \{v_1, \ldots, v_l, u_1, \ldots, u_m\}$,

(i) add the auxiliary variables $u_1, \ldots, u_m$ to $V$ (renaming if necessary so that none of them occurs before);

(ii) add the constraints $\langle (w_1^1, \ldots, w_{l_1}^1), \rho_1\rangle, \ldots, \langle (w_1^n, \ldots, w_{l_n}^n), \rho_n\rangle$ to $\mathcal{C}$;

(iii) remove $\langle s, \rho\rangle$ from $\mathcal{C}$.

It can easily be checked that the problem instance obtained by this procedure is equivalent to $(V; A; \mathcal{C})$ and belongs to $\mathbf{CSP}(\Gamma \cup \{=_A\})$. Moreover, since all the representations of relations from $\Delta$ are fixed, this transformation can be carried out in linear time in the size of the instance. Finally, all constraints of the form $\langle (v_1, v_2), =_A\rangle$ can be eliminated by replacing all occurrences of the variable $v_1$ with $v_2$. This transformation can also be carried out in polynomial time. $\square$

COROLLARY 2.17. *A set of relations* $\Gamma$ *is tractable if and only if the relational clone* $\langle\Gamma\rangle$ *is tractable.*

*Similarly,* $\Gamma$ *is* $\mathbf{NP}$*-complete if and only if* $\langle\Gamma\rangle$ *is* $\mathbf{NP}$*-complete.*

This result reduces the problem of characterizing tractable constraint languages to the problem of characterizing tractable relational clones.

*Example* 2.18. Reconsider the tractable constraint language of 0/1/all relations, $\Gamma_{\mathrm{ZOA}}$, defined in Example 2.13.

Note that for any fixed finite set $A$, the set of 0/1/all relations over $A$ contains only unary and binary relations and is therefore finite. However, it follows from Corollary 2.17 that the relational clone $\langle\Gamma_{\mathrm{ZOA}}\rangle$ is also tractable. This is an infinite set of relations containing relations of every possible arity.

In fact, the set $\langle \Gamma_{\mathrm{ZOA}} \rangle$ corresponds precisely to the *implicational* relations defined in [33]. Moreover, the set of 0/1/all relations, $\Gamma_{\mathrm{ZOA}}$, is precisely the set of unary and binary relations in the relational clone $\langle \Gamma_{\mathrm{ZOA}} \rangle$. $\square$

**2.4. From relational clones to sets of operations.** We have shown in the previous section that to analyze the complexity of arbitrary constraint languages it is sufficient to consider only relational clones. This considerably reduces the variety of languages to be studied. However, it immediately raises the question of how to represent and describe relational clones. For many relational clones the only known generating sets are rather sophisticated, and in some cases no generating sets are known [48].

Very conveniently, it turns out that there is a well-known alternative way to represent and describe any relational clone, using *operations*. In our definitions we follow [42] and [53].

Definition 2.19. *For any set $A$, and any natural number $n$, a mapping $f : A^n \to A$ is called an $n$-*ary operation *on $A$. The set of all finitary operations on $A$ is denoted by $O_A$.*

We first describe a fundamental algebraic relationship between operations and relations. Note that any operation on a set $A$ can be extended in a standard way to an operation on tuples of elements from $A$, as follows. For any ($m$-ary) operation $f$ and any collection of tuples $\overline{a}_1, \ldots, \overline{a}_m \in A^n$, where $\overline{a}_i = (a_{1i}, \ldots, a_{ni})$, define $f(\overline{a}_1, \ldots, \overline{a}_m)$ to be $(f(a_{11}, \ldots, a_{1m}), \ldots, f(a_{n1}, \ldots, a_{nm}))$.

Definition 2.20 (see [16, 48, 53]). *An $m$-ary operation $f \in O_A$* preserves *an $n$-ary relation $\rho \in R_A$ (or $f$ is a* polymorphism *of $\rho$, or $\rho$ is* invariant *under $f$) if $f(\overline{a}_1, \ldots, \overline{a}_m) \in \rho$ for all choices of $\overline{a}_1, \ldots, \overline{a}_m \in \rho$.*

For any given sets $\Gamma \subseteq R_A$ and $F \subseteq O_A$, let

$$\begin{aligned} \mathsf{Pol}(\Gamma) &= \{f \in O_A \mid f \text{ preserves each relation from } \Gamma\}, \\ \mathsf{Inv}(F) &= \{\rho \in R_A \mid \rho \text{ is invariant under each operation from } F\}. \end{aligned}$$

We remark that the operators $\mathsf{Pol}$ and $\mathsf{Inv}$ form a Galois correspondence between $R_A$ and $O_A$ (see Proposition 1.1.14 of [48]). Introductions to this correspondence can be found in [16, 47], and a comprehensive study in [48]. We note, in particular, that $\mathsf{Inv}(F) = \mathsf{Inv}(\mathsf{Pol}(\mathsf{Inv}(F)))$, for any set of operations $F$. Sets of the form $\mathsf{Inv}(F)$ are precisely the relational clones, as the next result indicates.

Proposition 2.21 (see [48]). *For any set $A$, and any $F \subseteq O_A$, the set $\mathsf{Inv}(F)$ is a relational clone. Conversely, any relational clone can be represented in the form $\mathsf{Inv}(F)$ for some set $F \subseteq O_A$. In particular, for any $\Gamma \subseteq R_A$, $\langle \Gamma \rangle = \mathsf{Inv}(\mathsf{Pol}(\Gamma))$.*

Using Proposition 2.21 together with Corollary 2.17, we can now translate our original problem of characterizing tractable sets of relations (Problem 2.12) into an equivalent problem for sets of operations. First, we define what it means for a set of operations to be tractable or **NP**-complete.

Definition 2.22. *A set $F \subseteq O_A$ is said to be* tractable *if $\mathsf{Inv}(F)$ is tractable. A set $F \subseteq O_A$ is said to be* **NP**-complete *if $\mathsf{Inv}(F)$ is* **NP***-complete.*

Using this definition, we obtain the following translation of Problem 2.12.

Problem 2.23 ("tractable operations problem"). *Characterize all tractable sets of operations on finite sets.*

In many cases the description of a set of operations provides a compact, concise way to describe the associated relational clone, as the next example indicates.

*Example* 2.24. Recall the constraint language $\Gamma_{\mathrm{ZOA}}$ of 0/1/all relations which was defined in Example 2.13 and extended to $\langle \Gamma_{\mathrm{ZOA}} \rangle$ in Example 2.18.

It was shown in [30] that $\langle \Gamma_{\text{ZOA}} \rangle$ is precisely the constraint language consisting of all relations which are invariant under the ternary "dual discriminator" operation $d$, defined as follows:

$$d(x, y, z) = \begin{cases} y & \text{if } y = z, \\ x & \text{otherwise.} \end{cases}$$

Hence this infinite tractable constraint language, which is rather complicated to describe in detail, may be represented very simply as the set of relations invariant under the tractable set of operations $\{d\}$. □

In many cases, it has been shown that the presence of a single operation satisfying certain simple conditions is sufficient to ensure the tractability of a set of operations.

*Example* 2.25. A binary operation $f(x, y)$ satisfying the following three conditions is said to be a semilattice operation:[1]

(i) $f(x, f(y, z)) = f(f(x, y), z)$ (associativity),
(ii) $f(x, y) = f(y, x)$ (commutativity),
(iii) $f(x, x) = x$ (idempotency).

Theorem 16 of [29] says that for any finite set $A$, any set of operations $F \subseteq O_A$ containing a semilattice operation is tractable. □

In contrast, we will now consider the properties of operations that are associated with **NP**-complete constraint languages.

DEFINITION 2.26. *An operation $f : A^n \to A$ is called* essentially unary *if there exists a (nonconstant) unary operation $g : A \to A$ and an index $i \in \{1, 2, \ldots, n\}$ such that $f(a_1, a_2, \ldots, a_n) = g(a_i)$ for all choices of $a_1, a_2, \ldots, a_n$. If $g$ is the identity operation, then $f$ is called a* projection.

*Any operation which is not essentially unary (including all constant operations) will be called* essentially nonunary.

PROPOSITION 2.27 (see Jeavons [27]). *For any finite set $A$ and any $\Gamma \subseteq R_A$, if $\mathsf{Pol}(\Gamma)$ contains essentially unary operations only, then* $\mathbf{CSP}(\Gamma)$ *is* **NP***-complete.*

*Example* 2.28. Consider the relation $N$ over the set $\{0, 1\}$, defined by

$$N = \{0, 1\}^3 \setminus \{(0, 0, 0), (1, 1, 1)\}.$$

It can be shown that $\mathsf{Pol}(\{N\})$ contains essentially unary operations only [49], and hence $\mathbf{CSP}(\{N\})$ is **NP**-complete, by Proposition 2.27.

We remark that the problem $\mathbf{CSP}(\{N\})$ was first shown to be **NP**-complete by Schaefer [52]; it corresponds to a restricted form of the NOT-ALL-EQUAL SATISFIABILITY problem [21, 45]. □

Boolean operations, that is, operations on $A = \{0, 1\}$, have been very well studied [49, 53]. In particular, it is known that if a Boolean constraint language is not contained in one of Schaefer's six tractable classes, then all of its polymorphisms are essentially unary operations [53]. Hence we may reformulate Schaefer's dichotomy theorem to obtain the following complete classification for Boolean operations.

COROLLARY 2.29 (Schaefer's dichotomy for operations). *A set of Boolean operations is tractable if it contains an essentially nonunary operation. Otherwise it is* **NP***-complete.*

[1]Note that in some earlier papers [27, 30, 46] the term *ACI operation* is used.

**3. Algebras.** We have shown in section 2 that the problem of analyzing the complexity of a constraint language can be translated into the problem of analyzing the complexity of the set of operations which preserve all of the relations in that language. In the Boolean case, this is sufficient to obtain a complete classification of complexity, but over larger sets we need to develop more powerful analytical tools, as the next example indicates.

*Example* 3.1. Consider the binary operation $\circ$ on the set $\{0, 1, 2\}$ defined by the following Cayley table:

| $\circ$ | 0 | 1 | 2 |
|---|---|---|---|
| 0 | 0 | 1 | 1 |
| 1 | 1 | 1 | 0 |
| 2 | 2 | 2 | 2 |

This operation does not fall into any known tractable class, nor is it essentially unary. Hence we cannot determine the complexity of this operation using the tools of the previous section (but see Example 5.5 below). □

In this section we shall open the way to the use of a further set of powerful analytical tools by making the final translation step, from sets of operations to *algebras.*

DEFINITION 3.2. *An* algebra *is an ordered pair* $\mathcal{A} = (A, F)$ *such that* $A$ *is a nonempty set and* $F$ *is a family of finitary operations on* $A$*. The set* $A$ *is called the* universe *of* $\mathcal{A}$*, and the operations from* $F$ *are called* basic*. An algebra with a finite universe is referred to as a* finite algebra.

To make the translation from sets of operations to algebras we simply note that any set of operations $F$ on a fixed set $A$ can be associated with the algebra $(A, F)$. Hence, we will define what it means for an algebra to be tractable by considering the tractability of the basic operations.

DEFINITION 3.3. *An algebra* $\mathcal{A} = (A, F)$ *is said to be* tractable *if* $F$ *is tractable.*
*An algebra* $\mathcal{A} = (A, F)$ *is said to be* **NP**-complete *if* $F$ *is* **NP**-*complete.*

Using Definition 3.3 we can now translate our original tractable relations problem (Problem 2.12) into the following equivalent problem for algebras.

PROBLEM 3.4 ("tractable algebras problem"). *Characterize all tractable algebras.*

Using Definition 3.3, we can reformulate Schaefer's dichotomy theorem [52] in yet another way, this time as a classification of the complexity of *algebras* defined on a two-element set.

COROLLARY 3.5 (Schaefer's dichotomy for algebras). *An algebra with a two-element universe is* **NP**-*complete if all of its basic operations are essentially unary. Otherwise it is tractable.*

The first advantage of using algebras instead of sets of operations is that we can make use of some standard constructions on algebras to obtain new results about the complexity of constraint languages. Another advantage is that finite algebras have been extensively studied, and a considerable body of structural theory has been developed [25, 42, 53]. We explore these ideas further in the remainder of the paper.

In our study it will be useful to describe an equivalence relation linking algebras that correspond to the same constraint language. As we noted earlier, the mappings $\mathsf{Pol}$ and $\mathsf{Inv}$ have the property that $\mathsf{Inv}(\mathsf{Pol}(\mathsf{Inv}(F))) = \mathsf{Inv}(F)$, and so we can extend a set of operations $F$ to the set $\mathsf{Pol}(\mathsf{Inv}(F))$ without changing the associated invariant relations. The set $\mathsf{Pol}(\mathsf{Inv}(F))$ consists of all operations that can be obtained from the operations in $F$, together with the set of all projection operations, by forming arbitrary compositions of operations [16, 48]. (If $f$ is an $n$-ary operation on $A$, and $g_1, g_2, \ldots, g_n$ are $k$-ary operations on $A$, then the *composition* of $f$ and $g_1, g_2, \ldots, g_n$ is the $k$-ary operation $h$ on $A$ defined by $h(a_1, a_2, \ldots, a_k) = f(g_1(a_1, \ldots, a_k), \ldots, g_n(a_1, \ldots, a_k))$.) The set of operations obtained in this way is usually referred to in universal algebra as the set of *term operations* over $F$ [16], so we will make the following definition.

DEFINITION 3.6. *For any algebra $\mathcal{A} = (A, F)$, an operation $f$ on $A$ will be called a* term operation *of $\mathcal{A}$ if $f \in \mathsf{Pol}(\mathsf{Inv}(F))$.*

*The set of all term operations of $\mathcal{A}$ will be denoted $\mathsf{Term}(\mathcal{A})$.*

Two algebras with the same universe are called *term equivalent* if they have the same set of term operations. Note that, for any algebra $\mathcal{A} = (A, F)$, we have $\mathsf{Inv}(F) = \mathsf{Inv}(\mathsf{Term}(\mathcal{A}))$, so two algebras are term equivalent if and only if they have the same set of associated invariant relations. It follows that we need to characterize tractable algebras only up to term equivalence.

*Example* 3.7. A *group* is an algebra with three basic operations: a binary multiplication operation, a unary converse operation, and a constant unit operation (see [37]). A *coset* of a group is a relation which is invariant under the ternary term operation $t(x, y, z) = xy^{-1}z$. It is stated in Theorem 33 of [20] that any constraint language consisting of cosets of a finite group is tractable. Hence any finite group is tractable, and moreover, any finite algebra with the ternary term operation $t$ is also tractable. $\square$

## 4. Special classes of algebras.

In this section we will show that, when studying the tractability of *finite* algebras, we can restrict our attention to certain special classes of algebras.

DEFINITION 4.1. *We call an algebra* surjective *if all of its term operations are surjective.*[2]

It is easy to verify that a finite algebra is surjective if and only if its unary term operations are all surjective and hence form a group of permutations.

It was shown in [27] that any unary polymorphism can be applied to a set of relations without changing the complexity of that set.

PROPOSITION 4.2 (see [27]). *For any set of relations $\Gamma$, and any unary operation $f \in \mathsf{Pol}(\Gamma)$, let $f(\Gamma)$ be the set of relations $\{f(\rho) \mid \rho \in \Gamma\}$, where $f(\rho) = \{f(\overline{a}) \mid \overline{a} \in \rho\}$. The set $\Gamma$ is tractable if and only if $f(\Gamma)$ is tractable, and $\Gamma$ is* **NP***-complete if and only if $f(\Gamma)$ is* **NP***-complete.*

Any algebra $\mathcal{A} = (A, F)$ which is not surjective will have a unary term operation $f$ which is not surjective, and hence has range $U$, where $U$ is a proper subset of $A$. By applying this operation to all of the relations in $\mathsf{Inv}(F)$, as described in Proposition 4.2, we can obtain a set of relations over $U$ without changing the tractability. The algebra corresponding to this new set of relations can be shown to be a *term induced* algebra of $\mathcal{A}$, which is defined as follows.

DEFINITION 4.3 (see [55]). *Let $\mathcal{A} = (A, F)$ be an algebra, and let $U$ be a nonempty subset of $A$. The* term induced algebra *$\mathcal{A}|_U$ is defined as $(U, \mathsf{Term}(\mathcal{A})|_U)$, where $\mathsf{Term}(\mathcal{A})|_U = \{g|_U : g \in \mathsf{Term}(\mathcal{A})$ and $g$ preserves $U\}$.*

By choosing a unary term operation $f$ with a range $U$ of *minimal* cardinality, we can ensure that the term induced algebra $\mathcal{A}|_U$ is surjective. Hence we have the following theorem.

THEOREM 4.4. *For any finite algebra $\mathcal{A}$, there exists a set $U$ such that*

(i) *the algebra $\mathcal{A}|_U$ is surjective, and*

[2]Note that in [54] an algebra is said to be surjective if all of its basic operations are surjective. However, such algebras can have nonsurjective term operations, so our definition is more restrictive.

(ii) *the algebra $\mathcal{A}$ is tractable if and only if $\mathcal{A}|_U$ is tractable, and is* **NP***-complete if and only if $\mathcal{A}|_U$ is* **NP***-complete.*

Theorem 4.4 shows that we can restrict our attention to surjective algebras. The next theorem shows that for many purposes we need consider only those surjective algebras with the additional property of being *idempotent*.

DEFINITION 4.5. *An operation $f$ on $A$ is called* idempotent *if it satisfies $f(x, \ldots, x) = x$ for all $x \in A$.*

*The* full idempotent reduct *of an algebra $\mathcal{A} = (A, F)$ is the algebra $(A, \mathsf{Term}_{id}(\mathcal{A}))$, where $\mathsf{Term}_{id}(\mathcal{A})$ consists of all idempotent operations from $\mathsf{Term}(\mathcal{A})$.*

Note that an operation $f$ on a set $A$ is idempotent if and only if it preserves all the relations in the set $\Gamma_{\text{CON}} = \{\{(a)\} \mid a \in A\}$, consisting of all unary one-element relations on $A$. Hence, $\mathsf{Inv}(\mathsf{Term}_{id}(\mathcal{A}))$ is the relational clone generated by $\mathsf{Inv}(F) \cup \Gamma_{\text{CON}}$.

To establish the next theorem we need an auxiliary lemma from [53].

LEMMA 4.6. *Let $\mathcal{A} = (\{a_1, a_2, \ldots, a_k\}, F)$ be a finite algebra whose unary term operations form a permutation group $G$. Then the relation $\rho_G$, defined by*

$$\rho_G = \{(g(a_1), \ldots, g(a_k)) \mid g \in G\},$$

*belongs to $\mathsf{Inv}(F)$.*

*Proof.* Proposition 1.3 of [53] states that relations of the form $\rho_G$ are preserved by all operations of the algebra $\mathcal{A}$ and hence belong to $\mathsf{Inv}(F)$. ◻

THEOREM 4.7. *A finite surjective algebra $\mathcal{A}$ is tractable if and only if its full idempotent reduct $\mathcal{A}_0$ is tractable. Moreover, $\mathcal{A}$ is* **NP***-complete if and only if $\mathcal{A}_0$ is* **NP***-complete.*

*Proof.* Let $\mathcal{A} = (A, F)$ be a finite surjective algebra, and let $\mathcal{A}_0$ be the full idempotent reduct of $\mathcal{A}$.

As observed above, $\mathsf{Inv}(\mathsf{Term}(\mathcal{A}_0))$ is the relational clone generated by the set $\mathsf{Inv}(F) \cup \Gamma_{\text{CON}}$, where $\Gamma_{\text{CON}} = \{\{(a)\} \mid a \in A\}$. By Corollary 2.17, it follows that $\mathcal{A}_0$ is tractable if and only if $\mathsf{Inv}(F) \cup \Gamma_{\text{CON}}$ is tractable, and $\mathcal{A}_0$ is **NP**-complete if and only if $\mathsf{Inv}(F) \cup \Gamma_{\text{CON}}$ is **NP**-complete. In the remainder of the proof we will show that $\mathbf{CSP}(\mathsf{Inv}(F) \cup \Gamma_{\text{CON}})$ is polynomial-time equivalent to $\mathbf{CSP}(\mathsf{Inv}(F))$.

Clearly, every instance of $\mathbf{CSP}(\mathsf{Inv}(F))$ may be considered as an instance of $\mathbf{CSP}(\mathsf{Inv}(F) \cup \Gamma_{\text{CON}})$, so there is a constant-time reduction from $\mathbf{CSP}(\mathsf{Inv}(F))$ to $\mathbf{CSP}(\mathsf{Inv}(F) \cup \Gamma_{\text{CON}})$.

For the converse result, let $\mathcal{P} = (V, A, \mathcal{C})$ be an instance of $\mathbf{CSP}(\mathsf{Inv}(F) \cup \Gamma_{\text{CON}})$ and let $\mathcal{P}'$ be the problem instance $(V', A, \mathcal{C}')$, where $V' = V \cup \{v_a \mid a \in A\}$ (each of the variables $v_a$ is a new variable not in $V$). To obtain the constraints $\mathcal{C}'$, take the original constraints $\mathcal{C}$ of $\mathcal{P}$, and replace each unary constraint of the form $\langle v, \{(a)\}\rangle$ in $\mathcal{C}$ with the constraint $\langle (v, v_a), =_A\rangle$, where $=_A$ is the binary equality relation on $A$. Finally, add the constraint $\langle (v_{a_1}, \ldots, v_{a_k}), \varrho_G\rangle$, where $a_1, a_2, \ldots, a_k$ are the elements of $A$ (in some order) and $\varrho_G$ is the relation defined in Lemma 4.6. Note that $\mathcal{P}'$ is an instance of $\mathbf{CSP}(\mathsf{Inv}(F))$ and that this construction can be carried out in polynomial time.

We claim that if the problem $\mathcal{P}'$ has a solution $\psi$, then it has a solution $\phi$ such that $\phi(v_a) = a$ for all $a \in A$. To establish this claim, note that, by the definition of $\rho_G$, there is $g \in G$ such that $\psi(v_a) = g(a)$ for all $a \in A$. Since $G$ is a group, the inverse operation $g^{-1} \in G$, which means that it is a term operation of $\mathcal{A}$. This implies that every relation in $\mathsf{Inv}(F)$ is invariant under $g^{-1}$, so $\phi = g^{-1}\psi$ is also a solution to $\mathcal{P}'$, and has the property that $g^{-1}\psi(v_a) = a$ for all $a \in A$.

Any solution $\phi$ satisfying this condition clearly satisfies all the constraints in $\mathcal{C}$, so the restriction of $\phi$ to $V$ is a solution to $\mathcal{P}$. Conversely, if $\psi$ is any solution to $\mathcal{P}$, then its extension $\psi'$ to $V'$ such that $\psi'(v_a) = a$, for all $a \in A$, is a solution to $\mathcal{P}'$. Hence this construction establishes a polynomial-time reduction from $\mathbf{CSP}(\mathsf{Inv}(F) \cup \Gamma_{\mathrm{CON}})$ to $\mathbf{CSP}(\mathsf{Inv}(F))$. $\square$

Theorem 4.7 can be restated in terms of constraint languages, as follows.

COROLLARY 4.8. *Let $\Gamma$ be a constraint language over a finite set $A$, and let $\Gamma_{\mathrm{CON}} = \{\{(a)\} \mid a \in A\}$ be the set of all unary one-element relations on $A$.*

*If all unary polymorphisms of $\Gamma$ are permutations, then $\Gamma$ is tractable if and only if $\Gamma \cup \Gamma_{\mathrm{CON}}$ is tractable, and $\Gamma$ is* **NP***-complete if and only if $\Gamma \cup \Gamma_{\mathrm{CON}}$ is* **NP***-complete.*

Corollary 4.8 has an interesting consequence connecting decision problems and search problems. In this paper we have formulated the constraint satisfaction problem as a *decision* problem (Definition 2.2), in which the question is to decide whether or not a solution exists. However, the corresponding *search* problem, in which the question is to *find* a solution, often arises in practice. We will now show that the tractable cases of these two forms of the problem coincide. (Note that the tractable cases of the search problem are those which belong to the complexity class **FP**.)

COROLLARY 4.9. *A decision problem $\mathbf{CSP}(\Gamma)$ is tractable if and only if the corresponding search problem can be solved in polynomial time.*

*Proof.* Obviously, tractability of the search problem implies tractability of the corresponding decision problem.

For the converse, let $\Gamma$ be a tractable set of relations over a finite set $A$. By choosing a unary polymorphism $f$ of $\Gamma$, whose image set $U$ is minimal, we can obtain a corresponding set of relations $\Gamma' = f(\Gamma)$ over $U$, such that every unary polymorphism of $\Gamma'$ is a permutation. By Proposition 4.2, $\Gamma'$ is also tractable.

Now consider any instance $\mathcal{P} = (V, A, \mathcal{C})$ of $\mathbf{CSP}(\Gamma)$. By the choice of $\Gamma$, we can decide in polynomial time in the size of $\mathcal{P}$ whether this instance has a solution. Assume that it has. Then the instance $\mathcal{P}' = (V, U, \mathcal{C}')$, obtained by replacing each constraint relation $\rho$ with the corresponding relation $f(\rho)$, also has a solution. Furthermore, every solution of $\mathcal{P}'$ is also a solution to $\mathcal{P}$.

Since $\mathcal{P}'$ has a solution, it follows that for each $v \in V$ there must be some $a \in A$ for which we can add the constraint $\langle (v), \{(a)\} \rangle$ and still have a solvable instance. Hence, by considering each variable in turn, and each possible value $a \in A$ for that variable, we can add such a constraint to each variable in turn, and hence obtain a solution to $\mathcal{P}'$. Checking for solvability for each possible value at each variable requires us to solve an instance of the decision problem $\mathbf{CSP}(\Gamma' \cup \Gamma_{\mathrm{CON}})$ at most $|V| \cdot |U|$ times, and hence can be completed in polynomial time in the size of $\mathcal{P}'$, by Corollary 4.8. $\square$

## 5. Constructions on algebras.

The results in this section link the complexity of a finite algebra with the complexity of its *subalgebras* and *homomorphic images* [9, 16, 42]. In many cases, we can use these results to reduce the problem of analyzing the complexity of an algebra to a similar problem involving an algebra with a smaller universe.

DEFINITION 5.1. *Let $\mathcal{A} = (A, F)$ be an algebra and $B$ a subset of $A$ such that, for any $f \in F$ and for any $b_1, \ldots, b_n \in B$, where $n$ is the arity of $f$, we have $f(b_1, \ldots, b_n) \in B$. Then the algebra $\mathcal{B} = (B, F|_B)$ is called a* subalgebra *of $\mathcal{A}$, where $F|_B$ consists of the restrictions of all operations in $F$ to $B$. If $B \neq A$, then $\mathcal{B}$ is said to be a* proper *subalgebra.*

THEOREM 5.2. *Let $\mathcal{A}$ be a finite algebra.*

(i) *If $\mathcal{A}$ is tractable, then so is every subalgebra of $\mathcal{A}$.*

(ii) *If $\mathcal{A}$ has an* **NP**-*complete subalgebra, then $\mathcal{A}$ is* **NP**-*complete.*

*Proof.* Let $\mathcal{B} = (B, F|_B)$ be a subalgebra of $\mathcal{A} = (A, F)$. It is easy to check that $\mathsf{Inv}(F|_B) \subseteq \mathsf{Inv}(F)$. Hence, $\mathbf{CSP}(\mathsf{Inv}(F|_B))$ can be reduced to $\mathbf{CSP}(\mathsf{Inv}(F))$ in constant time.

Now (i) and (ii) follow immediately from the existence of this reduction. ☐

DEFINITION 5.3. *Let $\mathcal{A}_1 = (A_1, F_1)$ and $\mathcal{A}_2 = (A_2, F_2)$ be such that $F_1 = \{f_i^1 \mid i \in I\}$ and $F_2 = \{f_i^2 \mid i \in I\}$, where both $f_i^1$ and $f_i^2$ are $n_i$-ary, for all $i \in I$.*

*A map $\varphi : A_1 \to A_2$ is called a* homomorphism *from $\mathcal{A}_1$ to $\mathcal{A}_2$ if*

$$\varphi f_i^1(a_1, \dots, a_{n_i}) = f_i^2(\varphi(a_1), \dots, \varphi(a_{n_i}))$$

*holds for all $i \in I$ and all $a_1, \dots, a_{n_i} \in A_1$.*

*If the map $\varphi$ is surjective, then $\mathcal{A}_2$ is said to be a* homomorphic image *of $\mathcal{A}_1$.*

THEOREM 5.4. *Let $\mathcal{A}$ be a finite algebra.*

(i) *If $\mathcal{A}$ is tractable, then so is every homomorphic image of $\mathcal{A}$.*

(ii) *If $\mathcal{A}$ has an* **NP**-*complete homomorphic image, then $\mathcal{A}$ is* **NP**-*complete.*

*Proof.* Let $\mathcal{B} = (B, F_B)$ be a homomorphic image of $\mathcal{A} = (A, F_A)$ and let $\varphi$ be the corresponding homomorphism. We will show that, for any finite $\Gamma \subseteq \mathsf{Inv}(F_B)$, $\mathbf{CSP}(\Gamma)$ is linear-time reducible to $\mathbf{CSP}(\Gamma')$ for some finite $\Gamma' \subseteq \mathsf{Inv}(F_A)$.

For $\rho \in \mathsf{Inv}(F_B)$, set $\varphi^{-1}(\rho) = \{\overline{a} \mid \varphi(\overline{a}) \in \rho\}$ where $\varphi$ acts componentwise. It is clear that $\varphi^{-1}(\rho)$ is a relation of the same arity as $\rho$. It can straightforwardly be checked that $\varphi^{-1}(\rho) \in \mathsf{Inv}(F_A)$. Let $\Gamma' = \{\varphi^{-1}(\rho) \mid \rho \in \Gamma\}$. Then $\Gamma'$ is a finite subset of $\mathsf{Inv}(F_A)$.

Take an instance $\mathcal{P} = (V, B, \mathcal{C})$ of $\mathbf{CSP}(\Gamma)$ and construct an instance $\mathcal{P}' = (V, A, \mathcal{C}')$ of $\mathbf{CSP}(\Gamma')$ where $\mathcal{C}' = \{\langle s, \varphi^{-1}(\rho)\rangle \mid \langle s, \rho\rangle \in \mathcal{C}\}$.

If $\psi$ is a solution of $\mathcal{P}'$, then $\varphi\psi$ is a solution of $\mathcal{P}$. Conversely, if $\xi$ is a solution of $\mathcal{P}$, then any function $\psi : V \to A$ such that $\varphi\psi(v) = \xi(v)$ for any $v \in V$ is a solution of $\mathcal{P}'$. ☐

We now give two examples to illustrate the use of Theorems 5.2 and 5.4. The examples show that *both* of these results can be useful (independently) to establish the complexity of certain algebras by reducing the question to an algebra over a smaller set.

*Example* 5.5. Reconsider Example 3.1. Let $\mathcal{A}$ be the idempotent algebra $(\{0, 1, 2\}, \circ)$, where $\circ$ is the binary operation defined by the following Cayley table:[3]

| $\circ$ | 0 | 1 | 2 |
|---|---|---|---|
| 0 | 0 | 1 | 1 |
| 1 | 1 | 1 | 0 |
| 2 | 2 | 2 | 2 |

By using Theorem 5.4, we will show that $\mathcal{A}$ is **NP**-complete even though all of its proper subalgebras are tractable.

Notice that, as the equalities $0 \circ 2 = 1$, $1 \circ 2 = 0$, $0 \circ 1 = 1 \circ 0 = 1$ show, $\mathcal{A}$ has only one proper subalgebra having more than one element, the algebra $\mathcal{B} = (\{0, 1\}, \circ|_{\{0,1\}})$. It is easy to check that $\circ|_{\{0,1\}}$ is a semilattice operation on $\{0, 1\}$. Hence, by Definition 3.3 and Theorem 16 of [29] (see Example 2.25), the algebra $\mathcal{B}$ is tractable.

[3]Note that we write $x \circ y$ instead of $\circ(x, y)$.

On the other hand, consider the algebra $\mathcal{C} = (C, *)$, where $C = \{a, b\}$ and for all $x, y \in \{a, b\}$, $x * y = x$. It is easy to check that the mapping $\varphi : \{0, 1, 2\} \to C$ such that $\varphi(0) = \varphi(1) = a$, $\varphi(2) = b$, is a homomorphism from $\mathcal{A}$ onto $\mathcal{C}$. By Corollary 3.5, $\mathcal{C}$ is **NP**-complete. Hence, by Theorem 5.4, $\mathcal{A}$ is **NP**-complete. ◻

*Example* 5.6. Consider the idempotent algebra $\mathcal{A} = (\{0, 1, 2\}, \circ)$, where $\circ$ is the binary operation defined by the following Cayley table:

| $\circ$ | 0 | 1 | 2 |
|---|---|---|---|
| 0 | 0 | 1 | 1 |
| 1 | 0 | 1 | 1 |
| 2 | 1 | 1 | 2 |

By using Theorem 5.2, we will show that $\mathcal{A}$ is **NP**-complete even though all of its smaller homomorphic images are tractable.

Since one-element algebras are certainly tractable, we need to consider only two-element homomorphic images $\mathcal{B}$ of $\mathcal{A}$. Let $\mathcal{B} = (\{a, b\}, *)$, where $*$ is a binary operation on $\{a, b\}$, and assume that $\varphi$ is a homomorphism from $\mathcal{A}$ onto $\mathcal{B}$. Then we have $\varphi(x \circ y) = \varphi(x) * \varphi(y)$ for all $x, y \in \{0, 1, 2\}$.

*Case* 1. $\varphi(0) = \varphi(1) = a$, $\varphi(2) = b$. In this case we have

$$\begin{aligned} a * a &= \varphi(0) * \varphi(0) = \varphi(0 \circ 0) = \varphi(0) = a, \\ a * b &= \varphi(0) * \varphi(2) = \varphi(0 \circ 2) = \varphi(1) = a, \\ b * a &= \varphi(2) * \varphi(0) = \varphi(2 \circ 0) = \varphi(1) = a, \\ b * b &= \varphi(2) * \varphi(2) = \varphi(2 \circ 2) = \varphi(2) = b. \end{aligned}$$

It is easy to check that $*$ is a semilattice operation on $\{a, b\}$. Hence, by Definition 3.3 and Theorem 16 of [29] (see Example 2.25), the algebra $\mathcal{B}$ is tractable.

*Case* 2. $\varphi(0) \neq \varphi(1)$.

It follows that $\varphi(2) \in \{\varphi(0), \varphi(1)\}$. If $\varphi(2) = \varphi(0)$, then

$$\varphi(0) = \varphi(0 \circ 0) = \varphi(0) * \varphi(0) = \varphi(0) * \varphi(2) = \varphi(0 \circ 2) = \varphi(1).$$

Alternatively, if $\varphi(2) = \varphi(1)$, then

$$\varphi(0) = \varphi(1 \circ 0) = \varphi(1) * \varphi(0) = \varphi(2) * \varphi(0) = \varphi(2 \circ 0) = \varphi(1).$$

Hence this second case is impossible, and we have shown that all smaller homomorphic images of $\mathcal{A}$ are tractable.

On the other hand, the algebra $\mathcal{A}$ has a subalgebra $\mathcal{A}' = (\{0, 1\}, \circ)$ such that $x \circ y = y$ for all $x, y \in \{0, 1\}$. By Corollary 3.5, $\mathcal{A}'$ is **NP**-complete. Hence, by Theorem 5.2, $\mathcal{A}$ is **NP**-complete. ◻

**6. Strictly simple surjective algebras.** The results of the previous section have established that a tractable algebra must have the property that all of its subalgebras and homomorphic images are tractable. Hence, a natural first question is whether we can classify the complexity of all algebras which do not have any smaller (nontrivial) subalgebras or homomorphic images. Classifying all such algebras can be viewed as a possible "base case for induction" in the pursuit of a general classification.

DEFINITION 6.1. *A finite algebra is called* simple *if all of its smaller homomorphic images are one-element; and* strictly simple[4] *if it is simple and all of its proper subalgebras are one-element.*

[4]In some papers appearing before 1990 such algebras are called *plain*.

By Theorem 4.4, it is sufficient to consider only surjective algebras. In this section we obtain a complete classification for all strictly simple surjective algebras.[5] We show that any algebra of this type is either tractable or **NP**-complete, and we give a complete characterization of the tractable cases. Such algebras include all surjective two-element algebras, as well as many algebras over larger universes, so this dichotomy result includes and generalizes Schaefer's dichotomy for algebras with a two-element universe (see Corollary 3.5 above).

To obtain the result, we make use of the complete description of finite strictly simple surjective algebras obtained by Szendrei [54]. To formulate Szendrei's result, we first need to introduce some further standard algebraic concepts and notation (for a general introduction to these algebraic concepts, see, for example, [37]).

Let $G$ be a permutation group acting on a set $A$. By $\mathcal{R}(G)$ we denote the set of operations on $A$ preserving each relation of the form $\{(a, g(a)) \mid a \in A\}$, for some $g \in G$. By $\mathcal{R}_{id}(G)$ we denote the set of idempotent operations in $\mathcal{R}(G)$.

A permutation group $G$ acting on a set $A$ is called *regular* if, for any $a, b \in A$, there exists $g \in G$ such that $g(a) = b$, and if each nonidentity member of $G$ has no fixed point. $G$ is called *primitive* if the algebra $(A, G)$ is simple.

Let $\overline{A} = (A, +)$ be a finite Abelian group, and let $K$ be a finite field. The finite dimensional vector space on $\overline{A}$ over $K$ will be denoted ${}_K\overline{A} = (A; +, K)$, the group of translations $\{x + a \mid a \in A\}$ will be denoted $T(\overline{A})$, and the endomorphism ring of ${}_K\overline{A}$ will be denoted $\mathsf{End}\ {}_K\overline{A}$. Note that one can consider $\overline{A}$ as a module over $\mathsf{End}\ {}_K\overline{A}$. This module will be denoted by ${}_{(\mathsf{End}\ {}_K\overline{A})}\overline{A}$.

Finally, let $\mathcal{F}_k^0$ denote the set of all operations preserving the relation

$$X_k^0 = \{(a_1, \dots, a_k) \in A^k \mid a_i = 0 \text{ for at least one } i,\ 1 \le i \le k\},$$

where 0 is some fixed element of $A$, and let $\mathcal{F}_\omega^0 = \bigcap_{k=2}^\infty \mathcal{F}_k^0$.

THEOREM 6.2 (see [54]). *Let $\mathcal{A}$ be a finite strictly simple surjective algebra.*

- *If $\mathcal{A}$ has no one-element subalgebras, then $\mathcal{A}$ is term equivalent to one of the following algebras:*
  - (a) *$(A, \mathcal{R}(G))$ for a regular permutation group $G$ acting on $A$;*
  - (b) *$(A, \mathsf{Term}_{\mathrm{id}}({}_{(\mathsf{End}\ {}_K\overline{A})}\overline{A}) \cup T(\overline{A}))$ for some vector space ${}_K\overline{A} = (A; +, K)$ over a finite field $K$;*
  - (c) *$(A, G)$ for a primitive permutation group $G$ on $A$.*
- *If $\mathcal{A}$ has one-element subalgebras, then $\mathcal{A}$ is idempotent and term equivalent to one of the following algebras:*
  - (a°) *$(A, \mathcal{R}_{id}(G))$ for a permutation group $G$ on $A$ such that every nonidentity member of $G$ has at most one fixed point;*
  - (b°) *$(A, \mathsf{Term}_{\mathrm{id}}({}_{(\mathsf{End}\ {}_K\overline{A})}\overline{A}))$ for some vector space ${}_K\overline{A}$ over a finite field $K$;*
  - (d) *$(A, \mathcal{R}_{id}(G) \cap \mathcal{F}_k^0)$ for some $k$ $(2 \le k \le \omega)$, some element $0 \in A$ and some permutation group $G$ acting on $A$ such that $0$ is the unique fixed point of every nonidentity member of $G$;*
  - (e) *$(A, F)$ where $|A| = 2$ and $F$ contains a semilattice operation;*
  - (f) *a two-element algebra with an empty set of basic operations.*

In the following theorem we determine all tractable finite strictly simple surjective algebras by analyzing each of the cases listed in Theorem 6.2.

[5]Note that the full idempotent reduct of a strictly simple surjective algebra is not always strictly simple. Hence we obtain a slightly stronger result by classifying all strictly simple surjective algebras, rather than just the strictly simple idempotent algebras.

THEOREM 6.3. *A finite strictly simple surjective algebra is* **NP***-complete if all of its term operations are essentially unary. Otherwise it is tractable.*

*Proof.* If $\mathcal{A}$ is an algebra of type (c) or (f), then all of its term operations are essentially unary. Hence $\mathsf{Pol}(\mathsf{Inv}(\mathsf{Term}(\mathcal{A})))$ contains essentially unary operations only, so $\mathcal{A}$ is **NP**-complete, by Proposition 2.27.

If $\mathcal{A} = (A, F)$ is an algebra of type (a) or (a°), then we claim that the dual discriminator operation $d(x, y, z)$ defined in Example 2.24 is a term operation of $\mathcal{A}$. To establish this claim, it is easy to verify that $d$ preserves every relation of the form $\{(a, g(a)) \mid a \in A\}$ where $g$ is a permutation on $A$. Since $d$ is an idempotent operation, it belongs to both $\mathcal{R}(G)$ and $\mathcal{R}_{id}(G)$. Hence, in cases (a) and (a°), every relation in $\mathsf{Inv}(F)$ is invariant under $d$. It follows from Theorem 5.7 of [30] that $\mathbf{CSP}(\mathsf{Inv}(F))$ is tractable (see Example 2.24). Hence, in this case $\mathcal{A}$ is tractable.

If $\mathcal{A}$ is an algebra of type (b) or (b°), then the affine operation $f(x, y, z) = x - y + z$ is a term operation of $\mathcal{A}$. Tractability of $\mathcal{A}$ then follows from Theorem 33 of [20] (see Example 3.7).

Now let $\mathcal{A} = (A, F)$ be an algebra of type (d) corresponding to some $k$ with $2 \leq k \leq \omega$. Consider the operation $f(x, y)$ defined as follows:

$$f(x,y) = \begin{cases} x & \text{if } x = y, \\ 0 & \text{otherwise.} \end{cases}$$

First, we show that $f$ preserves any relation of the form $g^\circ = \{(a, g(a)) \mid a \in A\}$ where $g \in G$. Let $\overline{a} = (a_1, a_2), \overline{b} = (b_1, b_2) \in g^\circ$. If $a_1 = b_1$, then $a_2 = g(a_1) = g(b_1) = b_2$, and, by definition of $f$, we have that the pair $f(\overline{a}, \overline{b})$ is equal to $(a_1, g(a_1))$ and hence belongs to $g^\circ$. If $a_1 \neq b_1$, then, since $g$ is a permutation, $a_2 \neq b_2$ and the pair $f(\overline{a}, \overline{b})$ equals $(0, 0)$, which also belongs to $g^\circ$ because 0 is a fixed point of $g$. Hence $f \in \mathcal{R}_{id}(G)$.

Next, we show that $f \in \mathcal{F}_k^0$. Let $\overline{a} = (a_1, \ldots, a_k), \overline{b} = (b_1, \ldots, b_k) \in X_k^0$; that is, each of $\overline{a}, \overline{b}$ contains 0. If, say, $a_i = 0$, then $f(a_i, b_i) = 0$, so the tuple $f(\overline{a}, \overline{b})$ contains 0 and hence belongs to $X_k^0$.

We have shown that $f \in \mathcal{R}_{id}(G) \cap \mathcal{F}_k^0$, and hence $f$ is a term operation of $\mathcal{A}$. It is easy to check that $f$ is a semilattice operation. By Theorem 16 of [29], this implies that $\mathbf{CSP}(\mathsf{Inv}(F))$ is tractable (see Example 2.25). Hence, in this case $\mathcal{A}$ is tractable.

Finally, if $\mathcal{A}$ is an algebra of type (e), then tractability again follows from Theorem 16 of [29]. ∎

## 7. Toward a general classification.

Theorem 6.3 gives a straightforward criterion to determine whether a finite strictly simple surjective algebra is tractable or **NP**-complete. In this section we examine what can be said about more general finite algebras.

Theorems 5.2 and 5.4 establish two separate necessary conditions for any finite algebra to be tractable. We can combine these conditions by using the standard algebraic notion of a *factor* [9].

DEFINITION 7.1. *A homomorphic image of a subalgebra of an algebra $\mathcal{A}$ is called a* factor *of $\mathcal{A}$. A factor whose universe contains only a single element is called a* trivial *factor.*

COROLLARY 7.2. *If $\mathcal{A}$ is a tractable finite algebra, then so is every factor of $\mathcal{A}$.*

Theorems 5.2 and 5.4 also establish two separate sufficient conditions for any finite algebra to be **NP**-complete. Using the notion of a factor, we can combine these results, together with Proposition 2.27, to obtain a single sufficient condition for **NP**-completeness.

Corollary 7.3. *A finite algebra $\mathcal{A}$ is* **NP**-*complete if it has a factor $\mathcal{B}$ all of whose term operations are essentially unary.*

However, the condition described in Corollary 7.3 is not a necessary condition for an arbitrary algebra $\mathcal{A}$ to be **NP**-complete, as the next example shows.

*Example* 7.4. Consider the algebra $\mathcal{A} = (\{0,1,2,3\}, \{d,f,p\})$, where $d$ is a binary operation and $f, p$ are unary operations, defined by the following tables:[6]

| $d$ | 0 | 1 | 2 | 3 |
|---|---|---|---|---|
| 0 | 0 | 3 | 0 | 3 |
| 1 | 2 | 1 | 2 | 1 |
| 2 | 2 | 1 | 2 | 1 |
| 3 | 0 | 3 | 0 | 3 |

| $x$ | $f(x)$ | $p(x)$ |
|---|---|---|
| 0 | 1 | 1 |
| 1 | 0 | 0 |
| 2 | 3 | 2 |
| 3 | 2 | 3 |

Since $f(0) = 1$, any subalgebra of $\mathcal{A}$ containing 0 also contains 1. Furthermore, since $f(1) = 0$, any subalgebra containing 1 also contains 0. Similarly, any subalgebra containing one of $2, 3$ also contains the other. Finally, since $d(0,1) = 3$ and $d(2,3) = 1$, it follows that the only subalgebra of $\mathcal{A}$ is $\mathcal{A}$ itself.

Now let $\phi$ be a homomorphism of $\mathcal{A}$. If $\phi(0) = \phi(1)$, then

$$\begin{aligned}\phi(2) &= \phi(d(1,0)) = d(\phi(1),\phi(0)) = d(\phi(0),\phi(0)) = \phi(0), \text{ and}\\ \phi(3) &= \phi(d(0,1)) = d(\phi(0),\phi(1)) = d(\phi(0),\phi(0)) = \phi(0),\end{aligned}$$

so $\phi$ maps all the elements of $\mathcal{A}$ to a single element. Furthermore, if $\phi(0) = \phi(2)$, then

$$\phi(0) = \phi(2) = \phi(p(2)) = p(\phi(2)) = p(\phi(0)) = \phi(p(0)) = \phi(1),$$

and we get the previous case. In all other cases a similar proof shows that if $\phi$ is not injective, then it maps all the elements of $\mathcal{A}$ to a single element.

Hence we have shown that the only nontrivial factor of $\mathcal{A}$ is $\mathcal{A}$ itself, and clearly not all the operations of $\mathcal{A}$ are essentially unary.

However, we will now show that $\mathcal{A}$ is **NP**-complete. (Note that this does not contradict Theorem 6.3 because $\mathcal{A}$ is not surjective.) To establish this, consider the ternary relation $\rho$, consisting of 36 tuples, defined as follows (where tuples are written as columns):

$$\rho = \begin{pmatrix} 0\,0\,0\,3\,3\,3\,0\,0\,0\,3\,3\,3\,1\,1\,1\,2\,2\,2\,1\,1\,1\,2\,2\,2\,1\,1\,1\,2\,2\,2\,0\,0\,0\,3\,3\,3 \\ 0\,3\,3\,0\,0\,3\,1\,1\,2\,1\,2\,2\,0\,0\,3\,0\,3\,3\,1\,2\,2\,1\,1\,2\,0\,0\,3\,0\,3\,3\,1\,1\,2\,1\,2\,2 \\ 1\,1\,2\,1\,2\,2\,0\,3\,3\,0\,0\,3\,0\,3\,0\,3\,0\,3\,0\,0\,3\,0\,3\,3\,1\,2\,2\,1\,1\,2\,1\,2\,1\,2\,1\,2 \end{pmatrix}.$$

It is straightforward to verify that this relation is invariant under the operations $d$, $f$, and $p$. However, if we set $h(x) = d(f(x), p(x))$, then $h(\rho) = N$, where $N$ is the relation defined in Example 2.28, and hence $\{h(\rho)\}$ is **NP**-complete. By Proposition 4.2, it follows that $\{\rho\}$ is **NP**-complete, and hence $\mathcal{A}$ is **NP**-complete. ☐

On the other hand, it was shown in Theorem 6.3 that the condition described in Corollary 7.3 is both necessary and sufficient for a finite strictly simple surjective algebra to be **NP**-complete (assuming that $\mathbf{P} \neq \mathbf{NP}$). Furthermore, all previously known forms of **NP**-complete constraint satisfaction problems (see [46, 52]) can be shown to be **NP**-complete using Corollary 7.3. Hence, we conjecture that the condition described in Corollary 7.3 is both necessary and sufficient for **NP**-completeness for any

[6]This algebra is, in fact, the *matrix square* [42] of $(\{0,1\}; ^{-})$, where $^{-}$ denotes the negation.

finite *surjective* algebra. We state this conjecture for the special case of idempotent algebras, where the only essentially unary operations are projections.

CONJECTURE 7.5 ("tractable algebras conjecture"). *A finite idempotent algebra $\mathcal{A}$ is* **NP***-complete if it has a nontrivial factor $\mathcal{B}$ all of whose operations are projections. Otherwise it is tractable.*

As shown in sections 2–4, the problem of determining the complexity of an arbitrary constraint language can be reduced to an equivalent problem for a certain idempotent algebra associated with the language. Therefore, this conjecture, if true, would solve all the various forms of the "tractability problem" mentioned above, including the original problem for arbitrary constraint languages, Problem 2.12.

The next examples show that Conjecture 7.5 is confirmed in a number of special cases, by existing dichotomy results. Moreover, we will show that in each case the existing dichotomy result can be obtained as a simple consequence of Conjecture 7.5.

*Example* 7.6. A dichotomy theorem for two-element algebras was given in Corollary 3.5. In this example we will show that this result is equivalent to Conjecture 7.5 restricted to two-element algebras.

Let $\mathcal{A}$ be a two-element algebra.

If $\mathcal{A}$ is idempotent, then Corollary 3.5 implies that either $\mathcal{A}$ is tractable, or else it is **NP**-complete and all of its operations are projections. Hence Corollary 3.5 implies that Conjecture 7.5 holds for any two-element algebra.

Conversely, we will now establish that Conjecture 7.5 implies Corollary 3.5. If $\mathcal{A}$ is not surjective, then it has a nonsurjective unary term operation, which must be constant, and hence $\mathcal{A}$ is tractable, by Theorem 4.4. On the other hand, if $\mathcal{A}$ is surjective, then by Theorem 4.7 we can consider its full idempotent reduct, $\mathcal{A}_0$. Assuming Conjecture 7.5 holds for any two-element algebra, we have that either $\mathcal{A}_0$ is tractable or else it is **NP**-complete and every operation of $\mathcal{A}_0$ is a projection. If every operation of $\mathcal{A}_0$ is a projection, then we claim that every operation of $\mathcal{A}$ must be essentially unary. To establish this claim, let $f$ be any term operation of $\mathcal{A}$. Since $\mathcal{A}$ is surjective, the unary term operation $g(x) = f(x, x, \ldots, x)$ must be a permutation. Now let $h$ be the composition of the inverse permutation $g^{-1}$ and $f$. It is easy to check that $h$ is an idempotent term operation of $\mathcal{A}$ and hence is an operation of $\mathcal{A}_0$. If $h$ is a projection, then $f$ must depend on only one of its arguments, which establishes the claim. Hence, we have shown that either $\mathcal{A}$ is tractable, or else every operation of $\mathcal{A}$ is essentially unary, which establishes that Conjecture 7.5 implies Corollary 3.5. □

*Example* 7.7. A dichotomy theorem for constraint languages on a three-element set was given as Theorem 4 of [3]. This result is stated in a very similar form to Conjecture 7.5, and is easily shown to be equivalent to this conjecture restricted to three-element algebras. □

*Example* 7.8. A constraint language containing all unary relations is called a *conservative* constraint language [5]. (One example of a problem with a conservative constraint language is the LIST $H$-COLORABILITY problem, defined in [19].)

It was shown in Theorem 4 of [5] that, for any conservative constraint language $\Gamma$ on a finite set $A$, either $\Gamma$ is tractable, or else there exists some two-element subset $B \subseteq A$, such that for every polymorphism $f$ of $\Gamma$, $f|_B$ is a projection. In this example we will show that this result is equivalent to Conjecture 7.5 restricted to algebras whose operations preserve all unary relations.

Let $\mathcal{A} = (A, F)$ be any finite algebra such that the set of relations $\Gamma = \mathsf{Inv}(F)$ contains all unary relations. Since $\Gamma$ contains every relation $\{(a)\}$, for each $a \in A$, the algebra $\mathcal{A}$ is idempotent. Furthermore, since $\Gamma$ contains every relation $\{(a_1), (a_2)\}$, for each two-element subset $B = \{a_1, a_2\} \subseteq A$, it follows that each of the corresponding algebras $\mathcal{B} = (B, F|_B)$ is a subalgebra of $\mathcal{A}$.

Hence, by Theorem 4 of [5], either $\Gamma$ is tractable, or else there exists some two-element subalgebra $\mathcal{B}$ of $\mathcal{A}$, all of whose operations are projections. Hence this result implies that Conjecture 7.5 holds for any algebra whose operations preserve all unary relations.

Conversely, to show that Conjecture 7.5 implies Theorem 4 of [5], assume that Conjecture 7.5 holds for the algebra $\mathcal{A}$. This implies that either $\Gamma$ is tractable, or else there exists some nontrivial factor $\mathcal{C}$ of $\mathcal{A}$ all of whose operations are projections. In the latter case, by the definition of a factor, $\mathcal{C}$ must be the image of some subalgebra $\mathcal{B}$ of $\mathcal{A}$ under some homomorphism $\varphi$. Choose elements $a_1, a_2 \in \mathcal{B}$ such that their images $\varphi(a_1), \varphi(a_2)$ in $\mathcal{C}$ are distinct. Since every two-element subset of $\mathcal{A}$ is a subalgebra, for any $f \in F$ we have that $f|_{\{a_1,a_2\}}$ has range $\{a_1, a_2\}$. Furthermore, the corresponding operation $f'$ of $\mathcal{C}$ is a projection, so we have that for any tuple $\overline{a}$ over $\{a_1, a_2\}$, the tuple $\varphi(f(\overline{a})) = f'(\varphi(\overline{a})) =$ the $i$th component of $\varphi(\overline{a})$ for some $i$. Hence $f|_{\{a_1,a_2\}}$ is also a projection, and we have shown that Conjecture 7.5 implies Theorem 4 of [5]. $\square$

Finally, we note that if Conjecture 7.5 is true, then it yields an effective procedure to determine whether any finite constraint language is tractable or **NP**-complete, as the following result indicates.

PROPOSITION 7.9. *Let $A$ be a finite set. If Conjecture 7.5 is true, then for any finite constraint language $\Gamma$ over $A$, it is possible to determine in polynomial time in the size of $\Gamma$ whether $\Gamma$ is* **NP***-complete or tractable.*

*Proof.* First set $\Gamma' = f(\Gamma) \cup \Gamma_{\text{CON}}$, where $f$ is a unary polymorphism of $\Gamma$ whose range $f(A) = U$ is minimal and $\Gamma_{\text{CON}} = \{\{(a)\} \mid a \in U\}$. (Note that the number of possible unary operations depends only on $|A|$, so $\Gamma'$ can be obtained in polynomial time in the size of $\Gamma$.) By Proposition 4.2 and Corollary 4.8, $\Gamma$ is **NP**-complete if and only if $\Gamma'$ is **NP**-complete, and $\Gamma$ is tractable if and only if $\Gamma'$ is tractable.

By Corollary 2.17 and Proposition 2.21, $\Gamma$ is **NP**-complete if and only if the idempotent algebra $\mathcal{A} = (U, Pol(\Gamma'))$ is **NP**-complete, and $\Gamma$ is tractable if and only if $\mathcal{A}$ is tractable. By Conjecture 7.5, $\mathcal{A}$ is **NP**-complete if it has a nontrivial factor $\mathcal{B}$ whose operations are all projections; otherwise it is tractable.

Assume first that $\mathcal{A}$ does have a nontrivial factor $\mathcal{B}$ whose operations are all projections, and let $\mathcal{B}$ be the homomorphic image of the subalgebra $\mathcal{A}'$ of $\mathcal{A}$ under the homomorphism $\varphi$. We may assume, without loss of generality, that the universe of $\mathcal{B}$ contains the set $\{0, 1\}$. The ternary **NP**-complete relation $N$, defined in Example 2.28, is preserved by all operations of $\mathcal{B}$, so the ternary relation $R = \varphi^{-1}(N)$ is preserved by all operations of $\mathcal{A}'$ and hence by all operations of $\mathcal{A}$. By Proposition 2.21, this implies that $R \in \langle \Gamma' \rangle$.

Conversely, assume that $R = \varphi^{-1}(N) \in \langle \Gamma' \rangle$, where $\varphi$ is an arbitrary unary function from some subset $A'$ of $U$ onto $\{0, 1\}$. By Proposition 2.21, $R$ is preserved by all operations of $\mathcal{A}$, and hence $A'$ is the universe of a subalgebra $\mathcal{A}'$ of $\mathcal{A}$. Furthermore, the relation $\varphi^{-1}(=_{\{0,1\}}) = \exists z R(x, z, z) \wedge R(y, z, z) \in \langle \Gamma' \rangle$. Using standard algebraic results (see Theorem 1.16 of [42]), this implies that $\varphi$ is a homomorphism from $\mathcal{A}'$ to a two-element factor $\mathcal{B}$ of $\mathcal{A}$ whose operations preserve $N$. Moreover, $\mathcal{B}$ is idempotent, so all its operations are projections.

It follows that $\Gamma$ is **NP**-complete if there is a relation $R \in \langle \Gamma' \rangle$ where $R$ is of the form $\varphi^{-1}(N)$ for some unary function $\varphi$ from some subset of $A$ onto $\{0, 1\}$, and in all other cases $\Gamma$ is tractable. By Proposition 1.1.19 of [48], the presence of any relation $R$ in $\langle \Gamma' \rangle$ can be determined by checking whether $R$ is preserved by all polymorphisms of $\Gamma'$ of arity bounded by $|R|$ (see [26] for an explicit construction). Since the number of possible unary functions $\varphi$ is less than $3^{|A|}$, and each $|R| \leq |A|^3$, this condition can be checked in polynomial time in the size of $\Gamma$, for any fixed finite set $A$. $\square$

**Acknowledgments.** The authors thank Victor Dalmau for reporting the result concerning idempotent reducts and for some comments simplifying the proof of Theorem 6.3, and David Cohen for drawing our attention to Corollary 4.9.

## REFERENCES

[1] W. Bibel, *Constraint satisfaction from a deductive viewpoint*, Artificial Intelligence, 35 (1988), pp. 401–413.

[2] M. Bodirsky and J. Nešetřil, *Constraint satisfaction with countable homogeneous templates*, in Proceedings of Computer Science Logic and the 8th Kurt Gödel Colloquium, Lecture Notes in Comput. Sci. 2803, Springer-Verlag, Berlin, 2003, pp. 44–57.

[3] A. Bulatov, *A dichotomy theorem for constraints on a three-element set*, in Proceedings of the 43rd IEEE Symposium on Foundations of Computer Science (FOCS'02), Vancouver, BC, Canada, 2002, pp. 649–658.

[4] A. Bulatov, *Mal'tsev Constraints Are Tractable*, Tech. report PRG-RR-02-05, Computing Laboratory, University of Oxford, Oxford, UK, 2002.

[5] A. Bulatov, *Tractable conservative constraint satisfaction problems*, in Proceedings of the 18th IEEE Symposium on Logic in Computer Science (LICS'03), Ottawa, ON, Canada, IEEE Press, 2003, pp. 321–330.

[6] A. Bulatov and P. Jeavons, *Tractable Constraints Closed under a Binary Operation*, Tech. report PRG-TR-12-00, Computing Laboratory, University of Oxford, Oxford, UK, 2002.

[7] A. Bulatov, A. Krokhin, and P. Jeavons, *The complexity of maximal constraint languages*, in Proceedings of the 33rd ACM Symposium on Theory of Computing (STOC'01), 2001, pp. 667–674.

[8] D. Cohen, P. Jeavons, P. Jonsson, and M. Koubarakis, *Building tractable disjunctive constraints*, J. ACM, 47 (2000), pp. 826–853.

[9] P. Cohn, *Universal Algebra*, Mathematics and Its Applications 6, D. Reidel, Dordrecht, The Netherlands, Boston, 1981.

[10] M. Cooper, D. Cohen, and P. Jeavons, *Characterising tractable constraints*, Artificial Intelligence, 65 (1994), pp. 347–361.

[11] N. Creignou, S. Khanna, and M. Sudan, *Complexity Classifications of Boolean Constraint Satisfaction Problems*, SIAM Monogr. Discrete Math. Appl. 7, SIAM, Philadelphia, 2001.

[12] V. Dalmau, *A new tractable class of constraint satisfaction problems*, in Proceedings of the 6th International Symposium on Artificial Intelligence and Mathematics, 2000.

[13] V. Dalmau, *Constraint satisfaction problems in non-deterministic logarithmic space*, in Proceedings of the 29th International Colloquium on Automata, Languages and Programming (ICALP'02), Lecture Notes in Comput. Sci. 2380, Springer-Verlag, Berlin, 2002, pp. 414–425.

[14] V. Dalmau and J. Pearson, *Set functions and width* 1 *problems*, in Proceedings of the 5th International Conference on Constraint Programming (CP'99), Lecture Notes in Comput. Sci. 1713, Springer-Verlag, Berlin, 1999, pp. 159–173.

[15] R. Dechter and J. Pearl, *Network-based heuristics for constraint satisfaction problems*, Artificial Intelligence, 34 (1988), pp. 1–38.

[16] K. Denecke and S. Wismath, *Universal Algebra and Applications in Theoretical Computer Science*, Chapman and Hall/CRC Press, Boca Raton, FL, 2002.

[17] T. Drakengren and P. Jonsson, *A complete classification of tractability in Allen's algebra relative to subsets of basic relations*, Artificial Intelligence, 106 (1998), pp. 205–219.

[18] T. Feder, *Constraint satisfaction on finite groups with near subgroups*, submitted; available online from http://theory.stanford.edu/~tomas/near.ps.

[19] T. Feder, P. Hell, and J. Huang, *List homomorphisms and circular arc graphs*, Combinatorica, 19 (1999), pp. 487–505.

[20] T. Feder and M. Vardi, *The computational structure of monotone monadic SNP and constraint satisfaction: A study through Datalog and group theory*, SIAM J. Comput., 28 (1998), pp. 57–104.

[21] M. Garey and D. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, Freeman, San Francisco, 1979.

[22] G. Gottlob, L. Leone, and F. Scarcello, *A comparison of structural CSP decomposition methods*, Artificial Intelligence, 124 (2000), pp. 243–282.

[23] M. Gyssens, P. Jeavons, and D. Cohen, *Decomposing constraint satisfaction problems using database techniques*, Artificial Intelligence, 66 (1994), pp. 57–89.

[24] P. Hell and J. Nešetřil, *On the complexity of $H$-coloring*, J. Combin. Theory Ser. B, 48 (1990), pp. 92–110.

[25] D. Hobby and R. McKenzie, *The Structure of Finite Algebras*, Contemp. Math. 76, AMS, Providence, RI, 1988.

[26] P. Jeavons, *Constructing constraints*, in Proceedings of the 4th International Conference on Principles and Practice of Constraint Programming, Lecture Notes in Comput. Sci. 1520, Springer-Verlag, London, 1998, pp. 2–16.

[27] P. Jeavons, *On the algebraic structure of combinatorial problems*, Theoret. Comput. Sci., 200 (1998), pp. 185–204.

[28] P. Jeavons, D. Cohen, and M. Cooper, *Constraints, consistency and closure*, Artificial Intelligence, 101 (1998), pp. 251–265.

[29] P. Jeavons, D. Cohen, and M. Gyssens, *A unifying framework for tractable constraints*, in Proceedings of the 1st International Conference on Constraint Programming (CP'95), Lecture Notes in Comput. Sci. 976, Springer-Verlag, 1995, pp. 276–291.

[30] P. Jeavons, D. Cohen, and M. Gyssens, *Closure properties of constraints*, J. ACM, 44 (1997), pp. 527–548.

[31] P. Jeavons, D. Cohen, and J. Pearson, *Constraints and universal algebra*, Ann. Math. Artificial Intelligence, 24 (1998), pp. 51–67.

[32] P. Jeavons and M. Cooper, *Tractable constraints on ordered domains*, Artificial Intelligence, 79 (1995), pp. 327–339.

[33] L. Kirousis, *Fast parallel constraint satisfaction*, Artificial Intelligence, 64 (1993), pp. 147–160.

[34] P. Kolaitis and M. Vardi, *Conjunctive-query containment and constraint satisfaction*, J. Comput. System Sci., 61 (2000), pp. 302–332.

[35] A. Krokhin, P. Jeavons, and P. Jonsson, *Reasoning about temporal relations: The tractable subalgebras of Allen's interval algebra*, J. ACM, 50 (2003), pp. 591–640.

[36] P. Ladkin and R. Maddux, *On binary constraint problems*, J. ACM, 41 (1994), pp. 435–469.

[37] S. Lang, *Algebra*, Grad. Texts in Math., Springer-Verlag, New York, 2002.

[38] D. Lesaint, N. Azarmi, R. Laithwaite, and P. Walker, *Engineering dynamic scheduler for work manager*, BT Technology J., 16 (1998), pp. 16–29.

[39] A. Mackworth, *Consistency in networks of relations*, Artificial Intelligence, 8 (1977), pp. 99–118.

[40] A. Mackworth, *Constraint satisfaction*, in Encyclopedia of Artificial Intelligence, Vol. 1, S. Shapiro, ed., Wiley Interscience, New York, 1992, pp. 285–293.

[41] A. Mackworth and E. Freuder, *The complexity of constraint satisfaction revisited*, Artificial Intelligence, 59 (1993), pp. 57–62.

[42] R. McKenzie, G. McNulty, and W. Taylor, *Algebras, Lattices and Varieties*, Vol. I, Wadsworth & Brooks/Cole Advanced Books & Software, Monterey, CA, 1987.

[43] U. Montanari, *Networks of constraints: Fundamental properties and applications to picture processing*, Inform. Sci., 7 (1974), pp. 95–132.

[44] B. Nebel and H.-J. Bürckert, *Reasoning about temporal relations: A maximal tractable subclass of Allen's interval algebra*, J. ACM, 42 (1995), pp. 43–66.

[45] C. Papadimitriou, *Computational Complexity*, Addison–Wesley, Reading, MA, 1994.

[46] J. Pearson and P. Jeavons, *A Survey of Tractable Constraint Satisfaction Problems*, Tech. report CSD-TR-97-15, Royal Holloway, University of London, London, 1997.

[47] N. Pippenger, *Theories of Computability*, Cambridge University Press, Cambridge, UK, 1997.

[48] R. Pöschel and L. Kalužnin, *Funktionen- und Relationenalgebren*, DVW, Berlin, 1979.

[49] E. Post, *The Two-Valued Iterative Systems of Mathematical Logic*, Annals Mathematics Studies 5, Princeton University Press, Princeton, NJ, 1941.

[50] L. Purvis and P. Jeavons, *Constraint tractability theory and its application to the product development process for a constraint-based scheduler*, in Proceedings of the 1st International Conference on the Practical Application of Constraint Technologies and Logic Programming (PACLP'99), The Practical Application Company, Blackpool, UK, 1999, pp. 63–79.

[51] J. Renz and B. Nebel, *On the complexity of qualitative spatial reasoning: A maximal tractable fragment of the region connection calculus*, Artificial Intelligence, 108 (1999), pp. 69–123.

[52] T. Schaefer, *The complexity of satisfiability problems*, in Proceedings of the 10th ACM Symposium on Theory of Computing (STOC'78), 1978, pp. 216–226.

[53] A. Szendrei, *Clones in Universal Algebra*, Séminaire de Mathematiques Supérieures 99, University of Montreal Press, Montreal, QC, Canada, 1986.

[54] A. Szendrei, *Simple surjective algebras having no proper subalgebras*, J. Austral. Math. Soc. Ser. A, 48 (1990), pp. 434–454.

[55] A. Szendrei, *Term minimal algebras*, Algebra Universalis, 32 (1994), pp. 439–477.

[56] P. van Beek and R. Dechter, *On the minimality and decomposability of row-convex constraint networks*, J. ACM, 42 (1995), pp. 543–561.

SIAM J. COMPUT.
Vol. 34, No. 3, pp. 743–762

# ONLINE SCHEDULING OF PRECEDENCE CONSTRAINED TASKS[*]

YUMEI HUO[†] AND JOSEPH Y.-T. LEUNG[†]

**Abstract.** A fundamental problem in scheduling theory is that of scheduling a set of $n$ tasks, with precedence constraints, on $m \geq 1$ identical and parallel processors so as to minimize the makespan (schedule length). In the past, research has focused on the setting whereby all tasks are available for processing at the beginning (i.e., at time $t = 0$). In this article we consider the situation where tasks, along with their precedence constraints, are released at different times, and the scheduler has to make scheduling decisions without knowledge of future releases. In other words, the scheduler has to schedule tasks in an online fashion. We consider both preemptive and nonpreemptive schedules. We show that *optimal* online algorithms exist for some cases, while for others it is impossible to have one. Our results give a sharp boundary delineating the possible and the impossible cases.

**Key words.** schedule length, preemptive and nonpreemptive scheduling, online and offline scheduling, parallel and identical processors, precedence constraints, intrees, outtrees

**AMS subject classifications.** Primary, 90B35; Secondary, 68C25

**DOI.** 10.1137/S0097539704444440

## 1. Introduction.

A fundamental problem in scheduling theory is that of scheduling a set of $n$ tasks, with precedence constraints, on $m \geq 1$ identical and parallel processors so as to minimize the makespan (schedule length). In this setting we are given $n$ tasks, $1, 2, \ldots, n$, where each task $j$ has a processing time $p_j$. The tasks have precedence constraints, $\prec$, in that $i \prec j$ signifies that task $j$ cannot start until task $i$ is finished. The tasks, together with their precedence constraints, are described by a directed acyclic graph $G = (V, A)$, where $V$ is a set of vertices representing the tasks and $A$ is a set of directed arcs representing the precedence constraints; there is a directed arc from task $i$ to task $j$ if $i \prec j$. We assume the graph has no transitive edges. The tasks can be scheduled preemptively or nonpreemptively. In preemptive scheduling, a task can be interrupted before it completes and later resumed on a possibly different processor. We assume that there is no time loss in preemption. By contrast, in nonpreemptive scheduling, a task, once begun, cannot be interrupted until it completes. With respect to a schedule $S$, the completion time of task $i$ is denoted by $C_i$, and the makespan is denoted by $C_{\max} = \max\{C_i\}$. The goal is to schedule the set of tasks on $m \geq 1$ identical and parallel processors so as to minimize $C_{\max}$. In the three-field classification scheme introduced by Graham et al. [4], the problems considered in this article are $P \mid prec \mid C_{\max}$ and $P \mid pmtn, prec \mid C_{\max}$.

A task $i$ is said to be an *immediate predecessor* of another task $j$ if there is a directed arc $(i, j)$ in $G$; $j$ is said to be an *immediate successor* of $i$. Task $i$ is said to be a *predecessor* of task $j$ if there is a directed *path* from $i$ to $j$ in $G$; $j$ is said to be a *successor* of $i$. We say that $G$ is an *intree* if each vertex, except the root, has exactly one immediate successor. $G$ is an *outtree* if each vertex, except the root, has exactly one immediate predecessor. A *chain* is an outtree in which each vertex has at most

[*]Received by the editors May 23, 2004; accepted for publication (in revised form) October 26, 2004; published electronically April 19, 2005. This work was supported in part by the National Science Foundation under grant DMI-0300156.

http://www.siam.org/journals/sicomp/34-3/44444.html

[†]Department of Computer Science, New Jersey Institute of Technology, Newark, NJ 07102 (yh23@njit.edu, leung@cis.njit.edu).

one immediate successor. We use *prec* to denote an arbitrary directed acyclic graph.

In the past, research in scheduling theory has concentrated on these four classes of precedence constraints: prec, intree, outtree, and chains. A number of polynomial-time algorithms have been developed. In nonpreemptive scheduling, the famous Coffman–Graham algorithm [1] is optimal for $P2 \mid p_j = 1, prec \mid C_{\max}$, while the well-known Hu algorithm [6] is optimal for $P \mid p_j = 1, intree \mid C_{\max}$ and $P \mid p_j = 1, outtree \mid C_{\max}$; we will describe these two algorithms in detail in the next section. It is known that $P \mid p_j = 1, prec \mid C_{\max}$ is strongly NP-hard [3], although the complexity is still open for each fixed $m \geq 3$. If the tasks have arbitrary processing times, then the problem becomes NP-hard in the ordinary sense even if we have two processors and the tasks are independent; i.e., $P2 \mid\mid C_{\max}$ is NP-hard in the ordinary sense [3].

For preemptive scheduling, the Muntz–Coffman algorithm [8, 9] is optimal for $P2 \mid pmtn, prec \mid C_{\max}$, $P \mid pmtn, intree \mid C_{\max}$, and $P \mid pmtn, outtree \mid C_{\max}$; we will describe this algorithm in detail in section 3. Again, $P \mid pmtn, prec \mid C_{\max}$ is strongly NP-hard [3], while the complexity is still open for each fixed $m \geq 3$.

All of the algorithms mentioned above assume that all tasks are available for processing at the beginning (i.e., at time $t = 0$). In this article, we consider the situation where tasks, along with their precedence constraints, are released at different times, and the scheduler has to make scheduling decisions without knowledge of future releases. In other words, the scheduler has to schedule tasks in an online fashion. We say that an online scheduling algorithm is *optimal* if it always produces a schedule with the minimum $C_{\max}$, i.e., a schedule as good as any schedule produced by any scheduling algorithm with full knowledge of future releases of tasks. Since an online scheduling algorithm has to schedule tasks in an online fashion, it is not clear that an optimal online scheduling algorithm necessarily exists. In this article, we show that online scheduling algorithms exist for some cases, while for others it is impossible to have one. Our results give a sharp boundary delineating the possible and the impossible cases.

We extend the notation of Graham et al. [4] to online scheduling problems in a natural way. For example, $P2 \mid p_j = 1, prec_i \textit{ released at } r_i \mid C_{\max}$ refers to the case where tasks with arbitrary precedence constraint, $prec_i$, are released at time $r_i$. In this case, there are two processors, each task has unit processing time, and preemption is not allowed. As another example, $P \mid pmtn, outtree_i \textit{ released at } r_i \mid C_{\max}$ refers to the case where tasks with outtree precedence constraint, $outtree_i$, are released at time $r_i$. In this case, there is an arbitrary number of processors, tasks have arbitrary processing times, and preemption is allowed.

Hong and Leung [5] have given an optimal online scheduling algorithm for a set of independent tasks on an arbitrary number of processors where preemption is allowed. The idea of their algorithm is to schedule tasks using a modified McNaughton wrap-around rule. (It is known that McNaughton's wrap-around rule is optimal for $P \mid pmtn \mid C_{\max}$ [7].) Tasks will be executed according to the schedule until new tasks arrive, at which time the algorithm will reschedule, by the same rule, the remaining portions of the unfinished tasks along with the newly arrived tasks. This process is repeated until all tasks are finished and no new tasks arrive.

Note that for nonpreemptive scheduling, it can be shown that it is impossible to have an optimal online algorithm for a set of independent tasks with arbitrary processing times, even if there are only two processors.

In this article, we show that optimal online scheduling algorithms exist for

(1) $P2 \mid p_j = 1, prec_i \textit{ released at } r_i \mid C_{\max}$;

(2) $P \mid p_j = 1, outtree_i \textit{ released at } r_i \mid C_{\max}$;

(3) $P2 \mid pmtn, prec_i\ released\ at\ r_i \mid C_{\max}$;
(4) $P \mid pmtn, outtree_i\ released\ at\ r_i \mid C_{\max}$.

Using an adversary argument, we show that it is impossible to have optimal online scheduling algorithms for

(1) $P3 \mid p_j = 1, intree_i\ released\ at\ r_i \mid C_{\max}$;
(2) $P2 \mid p_j = p, chains_i\ released\ at\ r_i \mid C_{\max}$;
(3) $P3 \mid pmtn, p_j = 1, intree_i\ released\ at\ r_i \mid C_{\max}$.

All of our optimal online scheduling algorithms follow the same format as the algorithm given in Hong and Leung [5]. For $P2 \mid p_j = 1, prec_i\ released\ at\ r_i \mid C_{\max}$, we use the Coffman–Graham algorithm to schedule tasks and follow the schedule to execute tasks until new tasks arrive, at which time we reschedule the remaining portions of the unfinished tasks along with the newly arrived tasks. We use Hu's algorithm for $P \mid p_j = 1, outtree_i\ released\ at\ r_i \mid C_{\max}$, and the Muntz–Coffman algorithm for $P2 \mid pmtn, prec_i\ released\ at\ r_i \mid C_{\max}$ and $P \mid pmtn, outtree_i\ released\ at\ r_i \mid C_{\max}$.

The organization of this article is as follows. In the next section we consider nonpreemptive scheduling, while preemptive scheduling will be considered in section 3. We draw some conclusions in the last section.

## 2. Nonpreemtive schedules.

In this section we consider nonpreemptive scheduling only. We first show that it is impossible to have optimal online algorithms for $P3 \mid p_j = 1, intree_i\ released\ at\ r_i \mid C_{\max}$ and $P2 \mid p_j = p, chains_i\ released\ at\ r_i \mid C_{\max}$. We then give an optimal online algorithm for $P2 \mid p_j = 1, prec_i\ released\ at\ r_i \mid C_{\max}$ in section 2.1 and an optimal online algorithm for $P \mid p_j = 1, outtree_i\ released\ at\ r_i \mid C_{\max}$ in section 2.2.

THEOREM 2.1. *It is impossible to have an optimal online algorithm for* $P3 \mid p_j = 1, intree_i\ released\ at\ r_i \mid C_{\max}$.

*Proof.* We will use an adversary argument to prove the theorem. Consider the intrees shown in Figure 2.1: the number of processors is three, $intree_1$ is released at $r_1 = 0$, and $intree_2$ is released at $r_2 = 4$.

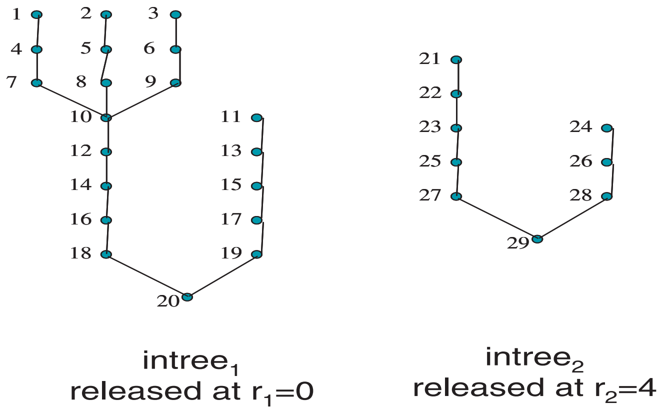
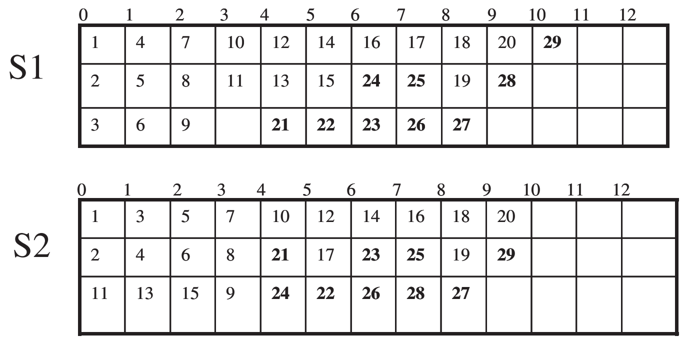


FIG. 2.1. *Example showing impossibility for* $P3|p_j = 1, intree_i\ released\ at\ r_i|C_{\max}$.

For $intree_1$, the length of the longest path is nine, so the makespan cannot be smaller than nine. To obtain the minimum makespan, task 10 must finish by time $t = 4$, which means that all its predecessors must be finished by time $t = 3$. Since task 10 has nine predecessors and since there are only three processors, all of the predecessors of task 10 must be executed in the first three time units. This means that there must be an idle processor in the time interval $[3, 4]$. Now, if $intree_2$ is released at time $t = 4$, then the makespan must be larger than 10. As shown in Figure 2.2, the optimal makespan is 10. On the other hand, if task 10 is not completed by time $t = 4$, then the schedule is already not optimal for $intree_1$.



| S1 | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | 1 | 4 | 7 | 10 | 12 | 14 | 16 | 17 | 18 | 20 | **29** | |
| | 2 | 5 | 8 | 11 | 13 | 15 | **24** | **25** | 19 | **28** | | |
| | 3 | 6 | 9 | | **21** | **22** | **23** | **26** | **27** | | | |

| S2 | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | 1 | 3 | 5 | 7 | 10 | 12 | 14 | 16 | 18 | 20 | | | |
| | 2 | 4 | 6 | 8 | **21** | 17 | **23** | **25** | 19 | **29** | | | |
| | 11 | 13 | 15 | 9 | **24** | **22** | **26** | **28** | **27** | | | | |

0
1
2
3
4
5
6
7
8
9
10
11
12

FIG. 2.2. *Schedule for the example in Figure* 2.1.

Thus, the adversary first releases $intree_1$ at time $t = 0$. If the online algorithm did not finish task 10 by time $t = 4$, then the schedule produced by the online algorithm is already not optimal for $intree_1$. On the other hand, if the online algorithm completes task 10 by time $t = 4$, then the adversary releases $intree_2$ at time $t = 4$. The online algorithm cannot finish both intrees by time $t = 10$, but the optimal makespan is 10. Again, the online algorithm did not produce an optimal schedule. $\square$

THEOREM 2.2. *It is impossible to have an optimal online algorithm for* $P2 \mid p_j = p, chains_i \text{ released at } r_i \mid C_{\max}$.

*Proof.* Consider the chains shown in Figure 2.3: two chains released at $r_1 = 0$, one chain released at $r_2 = 5$; each task in the chains has two units of processing time; and the number of processors is two. The minimum makespan for the first two chains (released at time $t = 0$) is 6, which can be attained only if both chains execute continuously from time $t = 0$ until time $t = 6$. Now, if the second chain is released at time $t = 5$, then the makespan will be 16. However, as shown in Figure 2.3, the optimal makespan is 15.

Thus, the adversary first releases the two chains at time $t = 0$. If the online algorithm leaves a processor idle in the time interval $[0, 5]$, then the schedule is already not optimal for the two chains. On the other hand, if the online algorithm keeps both processors busy during the interval $[0, 5]$, then the adversary releases the second chain at time $t = 5$. The online algorithm cannot finish all the chains by time 15, but the optimal makespan is 15. Again, the online algorithm did not produce an optimal schedule. $\square$

## 2.1. UET tasks, arbitrary precedence constraint, and two processors.

The Coffman–Graham algorithm is optimal for $P2 \mid p_j = 1, prec \mid C_{\max}$. It works by first assigning a label to each task which corresponds to the priority of the task; tasks with higher labels have higher priority. Once the labels are assigned, tasks are scheduled as follows: whenever a processor becomes free, assign it a task, all of whose predecessors have already been executed and which has the largest label among those tasks not yet assigned.

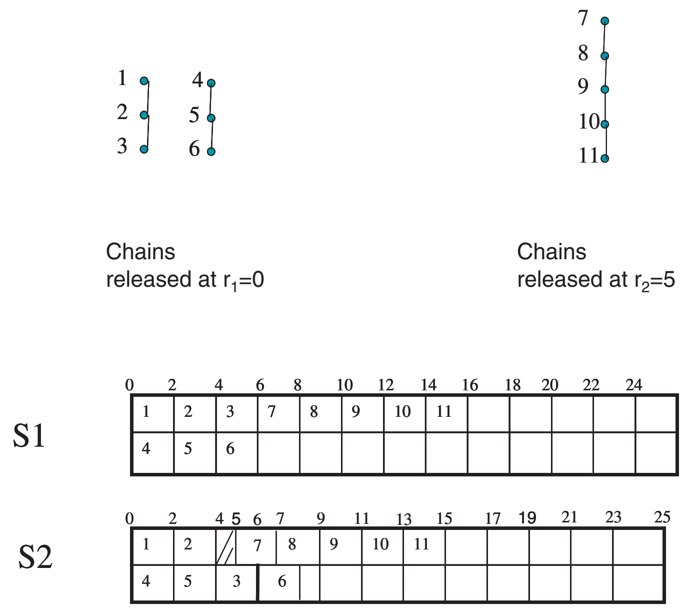


Fig. 2.3. *Example showing impossibility for* $P2 \mid p_j = p, chains_i$ *released at* $r_i \mid C_{\max}$.

Before we describe the labeling algorithm, we need to define a linear order on decreasing sequences of positive integers, as follows.

Definition 2.3. *Let* $N = (n_1, n_2, \ldots, n_t)$ *and* $N' = (n'_1, n'_2, \ldots, n'_{t'})$ *be two decreasing sequences of positive integers. We say that* $N < N'$ *if either of the following hold:*

1. *For some* $i$, $1 \le i \le t$, *we have* $n_j = n'_j$ *for all* $j$ *satisfying* $1 \le j \le i-1$ *and* $n_i < n'_i$.
2. $t < t'$ *and* $n_j = n'_j$ *for all* $j$ *satisfying* $1 \le j \le t$.

*Example.* $(8, 6, 4, 3) < (8, 6, 5)$ and $(9, 8, 6) < (9, 8, 6, 4, 3)$.

Let $n$ denote the number of tasks in *prec*. The labeling algorithm assigns to each task $i$ an integer label $\alpha(i) \in \{1, 2, \ldots, n\}$. The mapping $\alpha$ is defined as follows. Let $IS(i)$ denote the set of immediate successors of task $i$ and let $N(i)$ denote the decreasing sequence of integers formed by ordering the set $\{\alpha(j) \mid j \in IS(i)\}$.

1. An arbitrary task $i$ with $IS(i) = \emptyset$ is chosen and $\alpha(i)$ is defined to be 1.
2. Suppose for some $k \le n$ that the integers $1, 2, \ldots, k-1$ have been assigned. From the set of tasks for which $\alpha$ has been defined on all elements of their immediate successors, choose the task $j$ such that $N(j) \le N(i)$ for all such tasks $i$. Define $\alpha(j)$ to be $k$.
3. Repeat the assignment in step 2 until all tasks of *prec* have been assigned some integer.

*Example.* Figure 2.4 shows a set of tasks with their precedence constraints. The number inside each circle represents the task's index and the number next to each circle represents the label assigned to the task by the Coffman–Graham labeling algorithm. The schedule on two processors is also shown in Figure 2.4.

Our online algorithm utilizes the Coffman–Graham algorithm to schedule tasks. When new tasks arrive, the new tasks along with the unexecuted portion of the unfinished tasks will be rescheduled by the Coffman–Graham algorithm again.

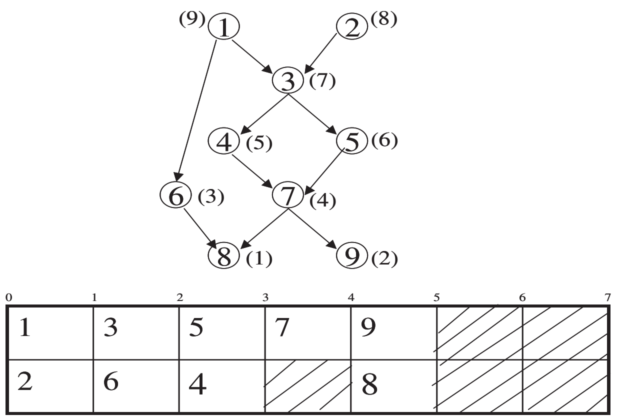


FIG. 2.4. *Example illustrating the Coffman–Graham algorithm.*

ALGORITHM A.

```
Whenever new tasks arrive, do {
    $t \leftarrow$ the current time;
    $U \leftarrow$ the set of tasks active (i.e., not finished) at time $t$;
    Call the Coffman–Graham algorithm to reschedule the tasks in $U$;
}
```

*Example.* Figure 2.5 shows another set of tasks released at time $r_2 = 2$ after the tasks in Figure 2.4 were released at time $r_1 = 0$. Note that tasks 4, 5, 7, 8, and 9 from the first release are unfinished at time $t = 2$. They are rescheduled, along with the new tasks from the second release, by the Coffman–Graham algorithm. The final schedule obtained by Algorithm A is also shown.

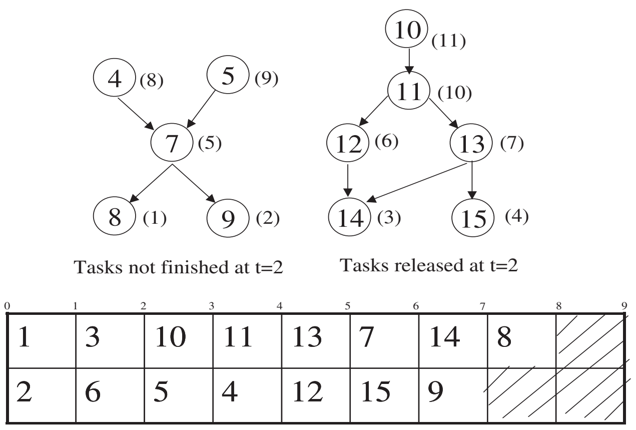


FIG. 2.5. *Example illustrating Algorithm* A.

The next lemma, whose proof will be omitted, is instrumental in proving that Algorithm A is optimal.

LEMMA 2.4. *Let S be a schedule for a set of tasks with arbitrary precedence constraints, where each task has unit processing time.*

1. *If S has the largest number of tasks completed at any time instant t, then S*

*must be optimal for $C_{\max}$.*

2. *If $S$ has the minimum idle processor time at any time instant $t$, then $S$ must be optimal for $C_{\max}$.*
3. *For two processors, the schedule $S'$ produced by the Coffman–Graham algorithm has the largest number of tasks completed at any time instant $t$.*

THEOREM 2.5. *Algorithm* A *is optimal for $P2 \mid p_j = 1, prec_i$ released at $r_i \mid C_{\max}$. Moreover, the schedule produced by Algorithm* A *has the largest number of tasks completed at any time instant $t$.*

*Proof.* We will prove the theorem by induction on the number, $i$, of release times. The basis case of $i = 1$ follows from Lemma 2.4. Assume the theorem is true for $i = k - 1$ release times; we will show that the theorem is true for $i = k$ release times.

Let $S_{k-1}$ denote the schedule obtained by Algorithm A after the first $k-1$ releases. By the induction hypothesis, $S_{k-1}$ is optimal for the tasks in the first $k - 1$ releases and has the largest number of tasks completed at any time instant $t$. The release time $r_k$ divides the tasks into two groups: (1) $\tau_1$, tasks completed by $r_k$ in $S_{k-1}$; and (2) $\tau_2$, tasks completed after $r_k$ in $S_{k-1}$. Let $S_k$ denote the schedule obtained by Algorithm A after the $k$th release. By the nature of Algorithm A, $S_k$ is identical to $S_{k-1}$ from time 0 until $r_k$. Thus, every task in $\tau_1$ is completed by $r_k$ in $S_k$ as well.

Let $\hat{S}_k$ be any schedule (including optimal schedules) for $k$ releases. The release time $r_k$ divides the tasks into two groups: (1) $\hat{\tau}_1$, tasks completed by $r_k$ in $\hat{S}_k$; and (2) $\hat{\tau}_2$, tasks completed after $r_k$ in $\hat{S}_k$. Let $prec_k$ denote all the tasks in the $k$th release. It is clear that the tasks in $\tau_1 \cup \tau_2$ are the same as the tasks in $\hat{\tau}_1 \cup \hat{\tau}_2 \setminus prec_k$.

We now construct another schedule $\bar{S}_k$ from $\hat{S}_k$ as follows: (1) Delete all the tasks in $\tau_1 \cup \tau_2$ from $\hat{S}_k$. (2) Schedule all the tasks in $\tau_1$ exactly as in $S_{k-1}$. The schedule $\bar{S}_k$ is identical to $S_{k-1}$ from time 0 until $r_k$. After $r_k$, it has tasks in $prec_k$ scheduled exactly as in $\hat{S}_k$ and idle processor times due to the deletion of the tasks in $\tau_1 \cup \tau_2$.

We want to show that we can schedule the tasks in $\tau_2$ into the idle processor times in $\bar{S}_k$ in such a way that the number of tasks completed at each time instant $t$ is not smaller than that in $\hat{S}_k$. By Lemma 2.4, $\bar{S}_k$ has the same makespan as $\hat{S}_k$.

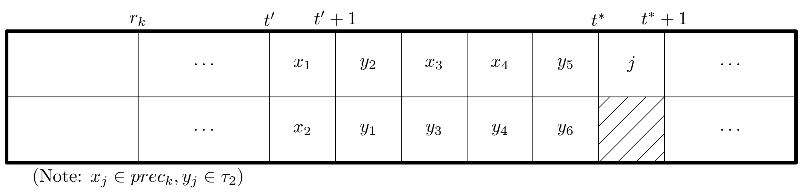
Let $L$ be the list of tasks in $\tau_2$ in ascending order of their completion times in $S_{k-1}$. We schedule the tasks in $\tau_2$ as follows. Whenever there is an idle processor time, scan the list $L$ and assign the first ready task encountered in the scan to the idle time.

We keep assigning tasks by the above method until we first encounter a time $t^*$ such that both processors are idle in the time interval $[t^*, t^* + 1]$, $j$ is the only task from $\tau_2$ that can be assigned in the interval, and the number of tasks completed by $t^* + 1$ in $\bar{S}_k$ is smaller than that completed at the same time in $\hat{S}_k$. Since we cannot assign another task in the interval, the remaining unassigned tasks in $\tau_2$ must all be successors of task $j$. There are two cases to consider.

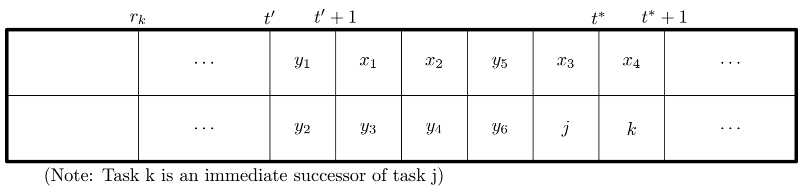
*Case* I. There is a time $t'$, $r_k \le t' < t^*$, such that both processors are executing some tasks in $prec_k$ in the time interval $[t', t' + 1]$. If there are several such times, let $t'$ be the largest.

Shown in Figure 2.6(a) is an example of Case I. In this figure, $x_j$ denotes a task in $prec_k$ and $y_j$ denotes a task in $\tau_2$. We transform the schedule to the one shown in Figure 2.6(b). After the transformation, task $j$ is completed by $t^*$ and an immediate successor of $j$ (task $k$ shown in the figure) can now be scheduled in the time interval $[t^*, t^* + 1]$. It is clear that there is no precedence constraint violation in the transformed schedule.

*Case* II. In every time interval $[t, t + 1]$, $r_k \le t < t^*$, at most one processor is


$r_k$
$t'$
$t'+1$
$t^*$
$t^*+1$
$\cdots$ $x_1$ $y_2$ $x_3$ $x_4$ $y_5$ $j$ $\cdots$
$\cdots$ $x_2$ $y_1$ $y_3$ $y_4$ $y_6$ $\cdots$
(Note: $x_j \in prec_k, y_j \in \tau_2$)

Fig(a) An example of Case I


$r_k$
$t'$
$t'+1$
$t^*$
$t^*+1$
$\cdots$ $y_1$ $x_1$ $x_2$ $y_5$ $x_3$ $x_4$ $\cdots$
$\cdots$ $y_2$ $y_3$ $y_4$ $y_6$ $j$ $k$ $\cdots$
(Note: Task k is an immediate successor of task j)

Fig(b) The transformed schedule

FIG. 2.6. *Example illustrating the proof of Case* I.

executing some task in $prec_k$.

In this case we pull out all the tasks in $prec_k$ that were scheduled in the time interval $[r_k, t^* + 1]$ and reschedule all the tasks in $\tau_2$ as in $S_{k-1}$. By the induction hypothesis, $S_{k-1}$ has the largest number of tasks completed at any time instant, and hence it can complete all the tasks by $t^*$. The schedule will be rearranged so that in every time interval $[t, t+1]$, $r_k \leq t < t^*$, at least one processor is executing a task in $\tau_2$; i.e., there is no time interval $[t, t+1]$ such that both processors are idle. We now schedule the tasks in $prec_k$ into the idle processor times and an immediate successor of $j$ in the time interval $[t^*, t^*+1]$. It is clear that the schedule has no precedence constraint violation.

After we perform the above operations, the number of tasks completed by time $t^* + 1$ in $\bar{S}_k$ is identical to that in $\hat{S}_k$. We can continue this operation until all tasks in $\tau_2$ have been scheduled. Thus, the number of tasks completed at each time instant $t$ is not smaller than that in $\hat{S}_k$.

Observe that $S_k$ is identical to $S_{k-1}$ from time 0 until $r_k$. Thus, it has the largest number of tasks completed at each time instant $t$ from time 0 up until $r_k$. After $r_k$, the tasks are scheduled by the Coffman–Graham algorithm, and hence $S_k$ has the largest number of tasks completed at each time instant $t$. ◻

**2.2. UET tasks, outtrees, and arbitrary number of processors.** Hu's algorithm is optimal for $P \mid p_j = 1, outtree \mid C_{\max}$. Like the Coffman–Graham algorithm, it first assigns a label to each task which corresponds to the priority of the task; tasks with higher labels have higher priority. Once the labels are assigned, tasks are scheduled as follows: whenever a processor becomes free, assign it a task, all of whose predecessors have already been executed and which has the largest label among those tasks not yet assigned. In Hu's algorithm, the label of a task is a function of the *level* of the task.

DEFINITION 2.6. *The level of a task i with no immediate successor is its processing time $p_i$. The level of a task with immediate successor(s) is its processing time plus the maximum level of its immediate successor(s).*

Hu's labeling algorithm assigns higher labels to tasks at higher levels; ties are broken in an arbitrary manner.

*Example.* Figure 2.7 shows a set of tasks with outtree precedence constraint. The number inside each circle is the task's index and the number next to each circle is the label given by Hu's algorithm. A schedule on four processors produced by Hu's algorithm is also shown.

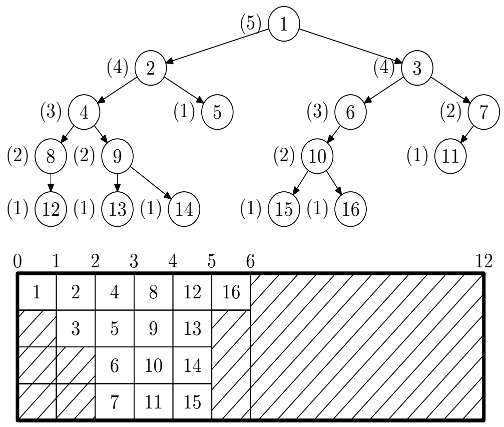


FIG. 2.7. *Example illustrating Hu's algorithm.*

Our online algorithm utilizes Hu's algorithm to schedule tasks. Whenever new tasks arrive, the new tasks along with the unexecuted portion of the unfinished tasks will be rescheduled by Hu's algorithm again.

ALGORITHM B.

```
Whenever new tasks arrive, do {
    $t \leftarrow$ the current time;
    $U \leftarrow$ the set of tasks active (i.e., not finished) at time $t$;
    Call Hu's algorithm to reschedule the tasks in $U$;
}
```

*Example.* Figure 2.8 shows another outtree released at time $r_2 = 3$ after the outtree shown in Figure 2.7 was released at time $r_1 = 0$. The schedule produced by Algorithm B is also shown.

THEOREM 2.7. *Algorithm* B *is optimal for* $P \mid p_j = 1, outtree_i$ *released at* $r_i \mid C_{\max}$. *Moreover, the schedule produced by Algorithm* B *has the largest number of tasks completed at any time instant* $t$.

*Proof.* Let $S$ be the schedule produced by Algorithm B for an instance of the $P \mid p_j = 1, outtree_i$ *released at* $r_i \mid C_{\max}$ problem and let $\hat{S}$ be any schedule (including optimal schedules). We will show, by contradiction, that the number of tasks completed in $S$ at each time instant $t$ is not smaller than that in $\hat{S}$. By Lemma 2.4, $S$ is an optimal schedule. Suppose this is not the case. Let $t'$ be the first time instant such that the number of tasks completed in $S$ is less than that in $\hat{S}$. Then there must be an idle processor in the time interval $[t' - 1, t']$ in $S$.

Consider the schedule $\hat{S}$. Let $k$ be the earliest completed task executed in $\hat{S}$ in the time interval $[0, t']$ that is not executed in $S$, and let $k$ be completed at time $t^*$ in $\hat{S}$. Assume that $k$ is in $outtree_i$. We claim that we can schedule $k$ in the time interval $[t' - 1, t']$ in $S$ as well. Suppose this is not the case. Then there must be a predecessor of $k$, say $j$, executing in the time interval $[t' - 1, t']$ in $S$. Let $t$ be the first time instant such that predecessors of $k$ are continuously executing from time $t$

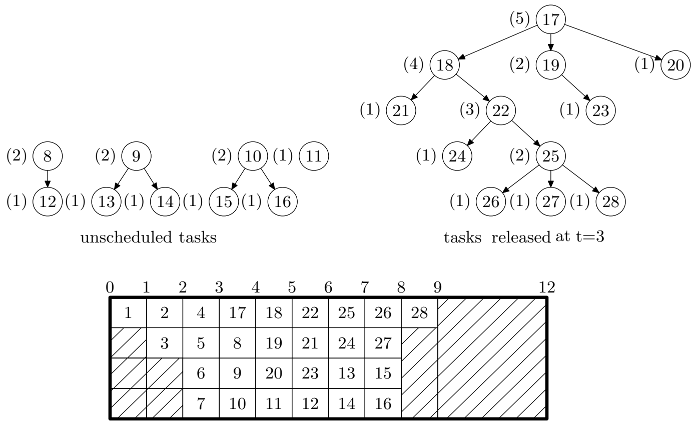


FIG. 2.8. *Example illustrating Algorithm* B.

until $t'$ in $S$, but that no predecessor of $k$ is executing in the time interval $[t-1, t]$. We have two cases to consider.

*Case* I. $t = r_i$.

In this case, it is clear that $k$ must be completed after $t'$ in any schedule whatsoever, contradicting our assumption that $k$ is completed by $t'$ in $\hat{S}$.

*Case* II. $t > r_i$.

Let $l$ be the predecessor of $k$ executed in the time interval $[t, t+1]$ in $S$. According to Hu's algorithm, $l$ was not executed in the time interval $[t-1, t]$ because the tasks executed in that time interval in $S$ all have levels greater than or equal to that of $l$, and because all processors are busy in the time interval. Since we are considering outtrees, every processor must be busy from time $t-1$ until $t'$, contradicting the fact that there is an idle processor in the time interval $[t'-1, t']$.

Repeating the above argument, we can show that the number of tasks completed by $t'$ in $S$ is not smaller than that in $\hat{S}$. $\square$

**3. Preemptive schedules.** In this section we consider preemptive scheduling only. We first show that it is impossible to have an optimal online algorithm for $P3 \mid pmtn, p_j = 1, intree_i \text{ released at } r_i \mid C_{\max}$. We then give an optimal online algorithm for $P2 \mid pmtn, prec_i \text{ released at } r_i \mid C_{\max}$ and $P \mid pmtn, outtree_i \text{ released at } r_i \mid C_{\max}$.

THEOREM 3.1. *It is impossible to have an optimal online algorithm for* $P3 \mid pmtn, p_j = 1, intree_i \text{ released at } r_i \mid C_{\max}$.

*Proof.* The proof given in Theorem 2.1 also proves this theorem since preemption won't help. $\square$

The Muntz–Coffman algorithm is optimal for the problems $P2 \mid pmtn, prec \mid C_{\max}$, $P \mid pmtn, intree \mid C_{\max}$, and $P \mid pmtn, outtree \mid C_{\max}$. It is essentially a highest-level-first strategy; see section 2.2 for the definition of level.

MUNTZ–COFFMAN ALGORITHM. Assign one processor each to the tasks at the highest level. If there is a tie among $y$ tasks (because they are at the same level) for the last $x$ $(x < y)$ processors, then assign $\frac{x}{y}$ processor to each of these $y$ tasks. Whenever either of the two events below occurs, reassign the processors to the unexecuted portion of the unfinished tasks according to the above rule. These are the following:

1. A task is completed.

2. We reach a point where, if we were to continue the present assignment, we would be executing some lower level tasks at a faster rate than other higher level tasks.

The schedule produced by the Muntz–Coffman algorithm is a processor-sharing schedule. We can convert it to a preemptive schedule by marking the time instants where processor assignment changes, and rescheduling the tasks executed between two adjacent time instants by McNaughton's wrap-around rule.

*Example.* Figure 3.1 shows a set of tasks with their precedence constraints. Inside each circle is the name of the task and its processing time. The number next to each circle is the level of the task. The processor-sharing schedule on two processors produced by the Muntz–Coffman algorithm is shown. Finally, the preemptive schedule constructed from the processor-sharing schedule is also shown.

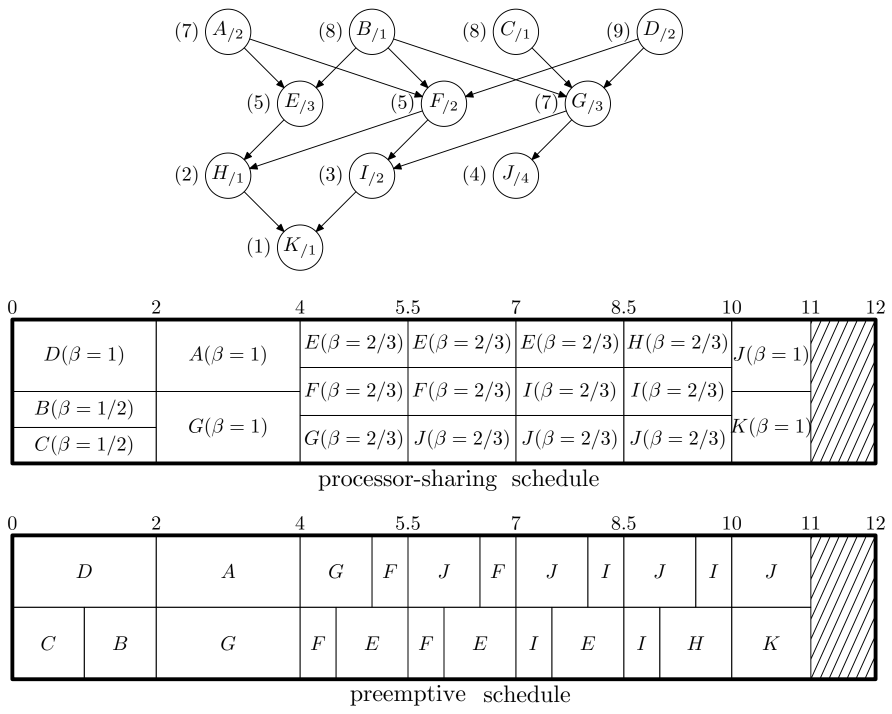


FIG. 3.1. *Example illustrating the Muntz–Coffman algorithm.*

Our online algorithm utilizes the Muntz–Coffman algorithm to schedule tasks. When new tasks arrive, the new tasks along with the unexecuted portion of the unfinished tasks will be rescheduled by the Muntz–Coffman algorithm again.

ALGORITHM C.

```
Whenever new tasks arrive, do {
    $t \leftarrow$ the current time;
    $U \leftarrow$ the set of tasks active (i.e., not finished) at time $t$;
    Call the Muntz–Coffman algorithm to reschedule the tasks in $U$;
}
```

Figure 3.2 shows another set of tasks released at time $r_2 = 6$ after the tasks in Figure 3.1 were released at time $r_1 = 0$. Algorithm C reschedules the unfinished portion of the unfinished tasks along with the new tasks. The processor-sharing

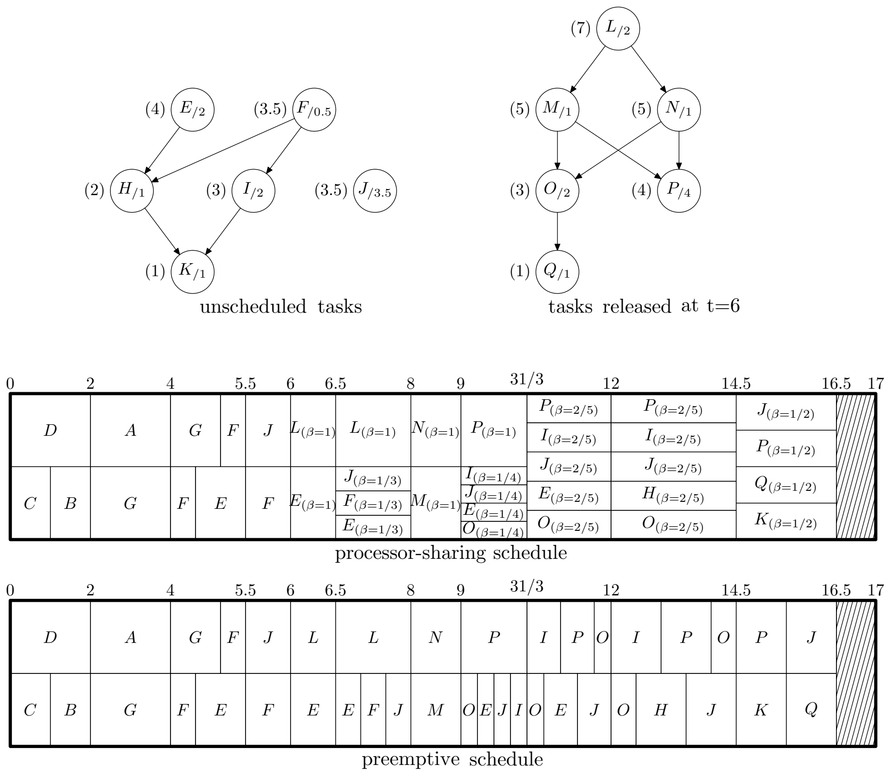


FIG. 3.2. *Example illustrating Algorithm* C.

schedule and the preemptive schedule are also shown.

Before we give the proofs that Algorithm C is optimal for $P2 \mid pmtn, prec_i$ *released at* $r_i \mid C_{\max}$ and $P \mid pmtn, outtree_i$ *released at* $r_i \mid C_{\max}$, we will give an optimal offline algorithm for these two cases.

ALGORITHM D. Assign one processor each to the tasks at the highest level. If there is a tie among $y$ tasks (because they are at the same level) for the last $x$ ($x < y$) processors, then assign $\frac{x}{y}$ processor to each of these $y$ tasks. Whenever one of the three events below occurs, reassign the processors to the unexecuted portion of the unfinished tasks according to the above rule. These are the following:

1. A task is completed.
2. We reach a point where, if we were to continue the present assignment, we would be executing some lower level tasks at a faster rate than other higher level tasks.
3. New tasks arrive.

Algorithm D is essentially the Muntz–Coffman algorithm, except that another new event—the arrival of new tasks—is added. In sections 3.1 and 3.2, we will prove that Algorithm D is an optimal offline algorithm for $P2 \mid pmtn, prec_i$ *released at* $r_i \mid C_{\max}$ and $P \mid pmtn, outtree_i$ *released at* $r_i \mid C_{\max}$, respectively. Shown in Figure 3.3 are the processor-sharing schedule and the preemptive schedule constructed by Algorithm D for the instance given in Figure 3.2.

Our proofs that Algorithm C is optimal for $P2 \mid pmtn, prec_i$ *released at* $r_i \mid C_{\max}$ and $P \mid pmtn, outtree_i$ *released at* $r_i \mid C_{\max}$ are based on the fact that Algorithm D

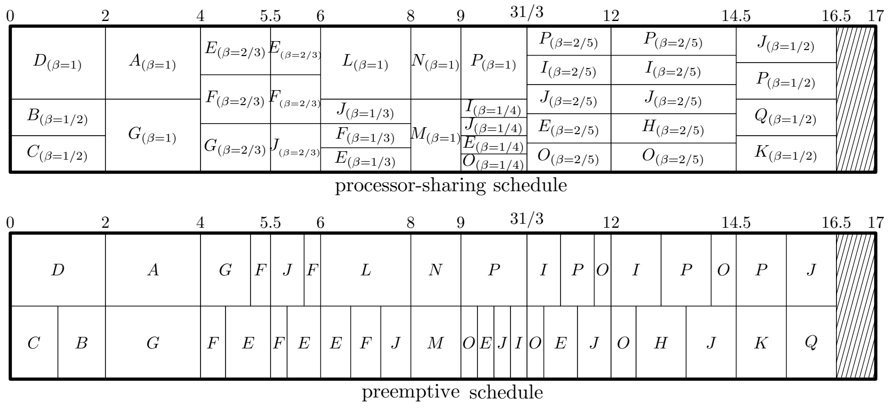

0 2 4 5.5 6 8 9 31/3 12 14.5 16.5 17
$D_{(\beta=1)}$ $A_{(\beta=1)}$ $E_{(\beta=2/3)}$ $E_{(\beta=2/3)}$ $L_{(\beta=1)}$ $N_{(\beta=1)}$ $P_{(\beta=1)}$ $P_{(\beta=2/5)}$ $P_{(\beta=2/5)}$ $J_{(\beta=1/2)}$
$I_{(\beta=2/5)}$ $I_{(\beta=2/5)}$ $P_{(\beta=1/2)}$
$F_{(\beta=2/3)}$ $F_{(\beta=2/3)}$ $J_{(\beta=2/5)}$ $J_{(\beta=2/5)}$
$B_{(\beta=1/2)}$ $G_{(\beta=1)}$ $J_{(\beta=1/3)}$ $M_{(\beta=1)}$ $I_{(\beta=1/4)}$ $J_{(\beta=1/4)}$ $E_{(\beta=2/5)}$ $H_{(\beta=2/5)}$ $Q_{(\beta=1/2)}$
$C_{(\beta=1/2)}$ $G_{(\beta=2/3)}$ $J_{(\beta=2/3)}$ $F_{(\beta=1/3)}$ $E_{(\beta=1/3)}$ $E_{(\beta=1/4)}$ $O_{(\beta=1/4)}$ $O_{(\beta=2/5)}$ $O_{(\beta=2/5)}$ $K_{(\beta=1/2)}$
processor-sharing schedule
0 2 4 5.5 6 8 9 31/3 12 14.5 16.5 17
D A G F J F L N P I P O I P O P J
C B G F E F E E F J M O E J I O E J O H J K Q
preemptive schedule

FIG. 3.3. *Example illustrating Algorithm* D.

is an optimal offline algorithm for these two cases.

**3.1. Arbitrary precedence constraint and two processors.** We first show that Algorithm D is an optimal offline algorithm for $P2 \mid pmtn, prec_i\ released\ at\ r_i \mid C_{\max}$, and then we show that Algorithm C is an optimal online algorithm for the same case.

LEMMA 3.2. *Let $S$ be the processor-sharing schedule produced by the Muntz–Coffman algorithm for an instance of $P2 \mid pmtn, prec \mid C_{max}$. Then $S$ has the minimum idle processor time at any time instant $t$.*

*Proof.* We will prove the lemma by contradiction. Let $S$ be the processor-sharing schedule produced by the Muntz–Coffman algorithm for an instance of $P2 \mid pmtn, prec \mid C_{max}$, and let $\hat{S}$ be any schedule (including optimal schedules) for the same instance. Let $t'$ be the first time instant such that the idle processor time in $S$ is larger than that in $\hat{S}$. Then there must be an idle processor in the time interval $[t', t' + \epsilon]$ in $S$ for some small positive number $\epsilon$. According to the Muntz–Coffman algorithm, the reason that a processor is idle in $[t', t' + \epsilon]$ is that no task is ready in the interval other than those that are already executing in the interval. Let $k$ be the earliest task executed in $\hat{S}$ in the time interval $[0, t' + \epsilon]$ but not in $S$, and let $k$ start its execution at time $t^*$ in $\hat{S}$. We assert that $k$ can be scheduled in the time interval $[t', t' + \epsilon]$ in $S$ as well. Suppose this is not the case. Then there must be a predecessor of $k$, say $j$, executing in the time interval $[t', t' + \epsilon]$ in $S$. Let $t$ be the first time instant such that predecessors of $k$ are continuously executing from time $t$ until $t' + \epsilon$ in $S$ but that no predecessor of $k$ is executing in the time interval $[t - \delta, t]$ for some small positive number $\delta$. We have two cases to consider.

*Case* I. $t = 0$.

If the predecessors of $k$ are continuously executed either by one full processor or without sharing any processors with jobs that are not predecessors of $k$ from time $t$ until $t' + \epsilon$, then it is clear that $k$ cannot be scheduled before $t' + \epsilon$ in any schedule whatsoever, contradicting the assumption that $k$ is scheduled by $t' + \epsilon$ in $\hat{S}$. On the other hand, if some predecessors of $k$ are sharing processors with a set of other tasks, say $U$, then there must be at least one ready task at time $t'$ that is either a task in $U$ or a successor of task(s) in $U$, and hence there will be no idle processor in $[t', t' + \epsilon]$, contradicting the assumption that there is at least one idle processor.

*Case* II. $t > 0$.

Let $l$ be the predecessor of $k$ executed at time $t$ in $S$. According to the Muntz–Coffman algorithm, $l$ was not executed in the time interval $[t-\delta, t]$ because the tasks executed in that interval all have levels greater than that of $l$ and all processors are busy in the time interval. Since the tasks in the time interval $[t-\delta, t]$ are not predecessors of $k$, there must be at least one job other than $l$ that is ready at time $t$. But this means that both processors are busy from time $t$ until $t' + \epsilon$, contradicting the fact that there is an idle processor in the time interval $[t', t' + \epsilon]$.

Repeating the above argument, we can show that the idle processor time in $S$ at each time instant $t$ is less than or equal to that in $\hat{S}$. □

Using the same technique as in Theorem 2.5 and the property given in Lemma 3.2, we can show that Algorithm D is an optimal offline algorithm for $P2 \mid pmtn, prec_i$ *released at* $r_i \mid C_{\max}$. This is stated in the next theorem, whose proof we will omit.

THEOREM 3.3. *Algorithm* D *is an optimal offline algorithm for* $P2 \mid pmtn, prec_i$ *released at* $r_i \mid C_{\max}$.

THEOREM 3.4. *Algorithm* C *is an optimal online algorithm for* $P2 \mid pmtn, prec_i$ *released at* $r_i \mid C_{\max}$.

*Proof.* Let $S$ be the schedule produced by Algorithm C for an instance of $P2 \mid pmtn, prec_i$ *released at* $r_i \mid C_{\max}$, and let $\hat{S}$ be the schedule produced by Algorithm D for the same instance. We will show, by induction on the number of release times, $i$, that $\hat{S}$ can be converted into $S$ without increasing the makespan and without violating any precedence constraints. Thus, $S$ is an optimal schedule as well.

The basis case, $i = 1$, is obvious, since $S$ and $\hat{S}$ are identical schedules. Assuming that the hypothesis is true for all $i \leq k-1$, we wish to show that the hypothesis is true for $i = k$. Let $t_1 < t_2 < \cdots < t_l$ be the time instants where event 1 or event 2 occurs in $\hat{S}$. Let $r_1$ and $r_2$ denote the first and second release times, respectively. There are two cases to consider.

*Case* I. $r_2 = t_j$ for some $1 \leq j \leq l$.

From time $r_1$ until $r_2$, $S$ is identical to $\hat{S}$. At time $r_2$, the remaining portions of the unfinished tasks in both schedules are identical. There are only $k-1$ releases from $r_2$ onward. By the induction hypothesis, $\hat{S}$ can be converted into $S$ without violating any precedence constraints.

*Case* II. $r_2$ is in the time interval $[t_j, t_{j+1}]$ for some $1 \leq j < l$.

There are two cases to consider.

*Case* II(a). Every task executing in the time interval $[t_j, t_{j+1}]$ is executing on a full processor.

The proof of this case is identical to that of Case I.

*Case* II(b). Some tasks are sharing processor(s) in the time interval $[t_j, t_{j+1}]$.

Let $x$ tasks be sharing $y$ processor(s) $(x > y)$ in the time interval $[t_j, t_{j+1}]$. Let $i_1, i_2, \ldots, i_x$ be the tasks sharing the $y$ processor(s). Since there are only two processors, there must be at most one task, say $j_1$, executing on a full processor. It is easy to see that from time $r_1$ until $t_j$, $S$ and $\hat{S}$ are identical schedules, but from $t_j$ until $r_2$, $S$ and $\hat{S}$ may not be the same.

The second release time, $r_2$, divides each task $i_1, i_2, \ldots, i_x$ in $S$ into two parts: those that were executed before $r_2$ and those that were executed after $r_2$. Let the level of a task, say $j$, at time $r_2$ be denoted by $level(j)$. We have $level(j_1) \geq level(i_l)$ for each $1 \leq l \leq x$, but the tasks $i_1, i_2, \ldots, i_x$ may have different levels. Consider now the schedule $\hat{S}$. We have $level(j_1) > level(i_1) = level(i_2) = \cdots = level(i_x)$. Note that $j_1$ may not exist if the tasks $i_1, i_2, \ldots, i_x$ share two processors.

We now convert the schedule $\hat{S}$ into one such that it is identical to $S$ in the interval

$[r_1, r_2]$ and such that the makespan is not increased and no precedence constraints are violated. From time $r_2$ and onward, there are $k-1$ releases. Thus, by the induction hypothesis, $\hat{S}$ can be converted into $S$ from time $r_2$ onward. Hence, $S$ is an optimal schedule as well.

From time $r_1$ until $t_j$, $\hat{S}$ and $S$ are identical schedules, but they may not be the same in the time interval $[t_j, r_2]$. We now show that we can convert $\hat{S}$ in the interval $[t_j, r_2]$ to be identical to $S$ in the same interval without increasing the makespan and without violating any precedence constraints. There are two cases to consider.

*Case* (i). Several tasks are sharing two processors in the time interval $[t_j, t_{j+1}]$.

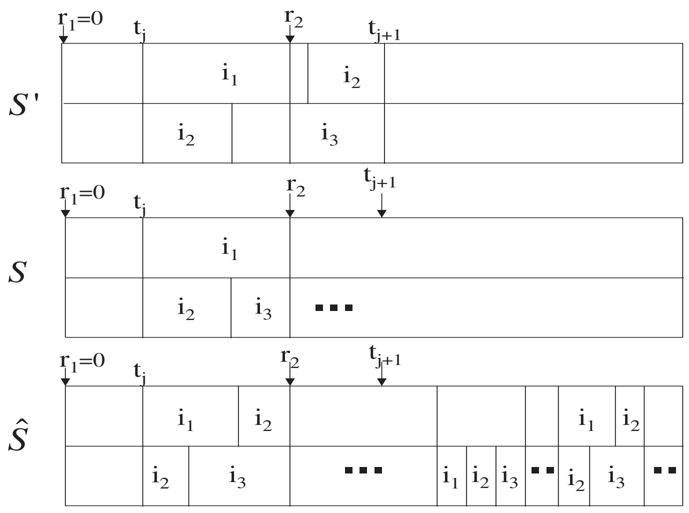


FIG. 3.4. *Example illustrating Case* (i).

An example of this case is shown in Figure 3.4: $S'$ is the schedule produced by Algorithm C before tasks were released at $r_2$, $S$ is the schedule obtained by Algorithm C after tasks were released at $r_2$, and $\hat{S}$ is the schedule produced by Algorithm D. We wish to show that the portions of the tasks scheduled in $S'$ in the interval $[r_2, t_{j+1}]$ can be rescheduled after time $r_2$ in $\hat{S}$ without increasing the makespan of $\hat{S}$ and without violating any precedence constraints.

Let $\hat{t}_1 < \hat{t}_2 < \cdots < \hat{t}_k$ be the time instants where event 1, 2, or 3 occurs in $\hat{S}$. We define $I_i$ to be the time interval $[\hat{t}_i, \hat{t}_{i+1}]$ in $\hat{S}$ for each $1 \le i < k$. Consider how the tasks $i_1$, $i_2$, and $i_3$ are scheduled after $r_2$ in $\hat{S}$. There are three cases to consider: (a) $i_1$, $i_2$, and $i_3$ share one processor in the intervals $I_{i_1}, I_{i_2}, \ldots, I_{i_p}$, but they are not scheduled in any other intervals; (b) $i_1$, $i_2$, and $i_3$ share two processors in the intervals $I_{j_1}, I_{j_2}, \ldots, I_{j_q}$, but they are not scheduled in any other intervals; and (c) $i_1$, $i_2$, and $i_3$ share one processor in the intervals $I_{i_1}, I_{i_2}, \ldots, I_{i_p}$ and two processors with other tasks in the intervals $I_{j_1}, I_{j_2}, \ldots, I_{j_q}$, and they are not scheduled in any other intervals.

In case (a), it is clear that we can schedule the tasks in the interval $[r_2, t_{j+1}]$ in $S'$ into the intervals $I_{i_1}, I_{i_2}, \ldots, I_{i_p}$ in $\hat{S}$ without increasing the makespan and without violating any precedence constraints. In case (b), we can divide the schedule in the interval $[r_2, t_{j+1}]$ in $S'$ into $q$ subintervals so that each subinterval will be scheduled into one of the intervals $I_{j_x}$, $1 \le x \le q$, in $\hat{S}$. Again, this will not increase the makespan or violate any precedence constraints. In case (c), let $l_0$ be the total length of all the intervals $I_{i_1}, I_{i_2}, \ldots, I_{i_p}$, and let $l_x$, $1 \leq x \leq q$, be the total execution time of the tasks $i_1$, $i_2$, and $i_3$ scheduled in $I_{j_x}$. We can schedule the tasks in the interval $[r_2, r_2 + \frac{l_0}{2}]$ in $S'$ into the intervals $I_{i_1}, I_{i_2}, \ldots, I_{i_p}$ in $\hat{S}$. We then divide the interval $[r_2 + \frac{l_0}{2}, t_{j+1}]$ in $S'$ into $q$ subintervals: the $x$th subinterval, $1 \leq x \leq q$, has length $\frac{l_x}{2}$. We take the tasks executed in $S'$ in the $x$th subinterval and schedule them in $I_{j_x}$ in $\hat{S}$, along with the other tasks executed in the same interval. It is easy to see that we can reschedule without increasing the makespan or violating any precedence constraints.

Using this approach, we can always convert $\hat{S}$ in $[t_j, r_2]$ to be identical to $S$ in the same interval without increasing the makespan of $\hat{S}$ and without violating any precedence constraints, no matter how many jobs are sharing the two processors in the time interval $[t_j, t_{j+1}]$.

*Case* (ii). One task is executing on a full processor while other tasks are sharing one processor in the time interval $[t_j, t_{j+1}]$.

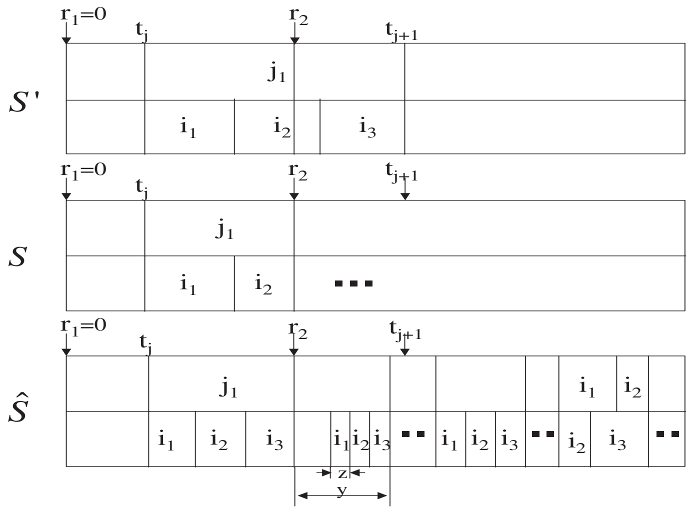


FIG. 3.5. *Example illustrating Case* (ii).

An example of this case is shown in Figure 3.5: $S'$ is the schedule constructed by Algorithm C before new tasks were released at $r_2$, $S$ is the schedule constructed by Algorithm C after new tasks were released at $r_2$, and $\hat{S}$ is the schedule produced by Algorithm D. Again, we want to show that the portions of the tasks scheduled in $S'$ in the interval $[r_2, t_{j+1}]$ can be rescheduled after time $r_2$ in $\hat{S}$ without increasing the makespan of $\hat{S}$ and without violating any precedence constraints.

As before, let $\hat{t}_1 < \hat{t}_2 < \cdots < \hat{t}_k$ be the time instants where event 1, 2, or 3 occurs in $\hat{S}$. We define $I_i$ to be the time interval $[\hat{t}_i, \hat{t}_{i+1}]$ in $\hat{S}$ for each $1 \leq i < k$. Consider how the tasks $i_1$, $i_2$, and $i_3$ are scheduled after $r_2$ in $\hat{S}$. There are two cases to consider: (a) $i_1$, $i_2$, and $i_3$ share one processor in the intervals $I_{i_1}, I_{i_2}, \ldots, I_{i_p}$, but they are not scheduled in any other intervals; (b) $i_1$, $i_2$, and $i_3$ share one processor in the intervals $I_{i_1}, I_{i_2}, \ldots, I_{i_p}$ and two processors with other tasks in the intervals $I_{j_1}, I_{j_2}, \ldots, I_{j_q}$, and they are not scheduled in any other intervals.

In case (a), it is clear that we can schedule the tasks on the second processor in $S'$ in the interval $[r_2, t_{j+1}]$ into the intervals $I_{i_1}, I_{i_2}, \ldots, I_{i_p}$ in $\hat{S}$ without increasing the makespan of $\hat{S}$ and without violating any precedence constraints. In case (b), let $y = t_{j+1} - r_2$. Assume that in the time interval $[r_2, t_{j+1}]$, $i_1, i_2, i_3$ have each executed $z$ units in $\hat{S}$, $0 \leq z \leq \frac{y}{3}$. This means that $y - 3z$ units of processor time are used by the newly released tasks in the same interval. It is easy to see that in $\hat{S}$, the remaining portions of $i_1, i_2, i_3$ at time $t_{j+1}$ are all $\frac{y}{3} - z$. On the other hand, the lengths of the remaining portions of $i_1, i_2, i_3$ in $S'$ in the interval $[r_2, t_{j+1}]$ are all different. Thus, after $r_2$, when we reschedule the tasks $i_1, i_2, i_3$ into the intervals $I_{j_1}, I_{j_2}, \ldots, I_{j_q}$, it is quite possible that we create overlaps. It is clear that the total length of all the overlaps is no more than $\frac{y-3z}{2}$.

Assume that both processors are used by a task, say $i_3$, in the time interval $[\lambda_1, \lambda_2]$. From $\lambda_1$ back to $r_2$, we use the following method to eliminate the overlap: Find the first time interval $[t^*, t^*+\epsilon]$ such that (1) one processor is used by $i_1, i_2, i_3, j_1$, or the successors of $j_1$, and the other processor, say the second processor, is used by other jobs; or (2) both processors are not used by $i_1, i_2, i_3, j_1$ or the successors of $j_1$. If (1) holds, then we can interchange the schedule on the second processor in $[t^*, t^* + \epsilon]$ with the schedule on the second processor in $[\lambda_2 - \epsilon, \lambda_2]$, so the overlap is reduced by $\epsilon$. (Note that if $i_3$ is executing in the interval $[t^*, t^* + \epsilon]$, the interchange does not reduce the overlap, but it has the effect of pushing backwards the time where the overlap occurs.) If (2) holds, then we interchange the schedule on the second processor in $[t^*, t^* + \epsilon]$ with the schedule on the second processor in $[\lambda_2 - \epsilon, \lambda_2]$, and we interchange the schedule on the first processor in $[t^*, t^* + \epsilon]$ with the schedule on the second processor in $[\lambda_2 - 2 * \epsilon, \lambda_2 - \epsilon]$. Again, the overlap is reduced by $\epsilon$ and no precedence constraints are violated.

We repeat the above operation until all of the overlaps are eliminated. Since in the time interval $[r_2, t_{j+1}]$, one processor has $y - 3z$ processor time used by the newly released tasks, we can always eliminate the overlap. Thus, we can convert the schedule in $[t_j, r_2]$ in $\hat{S}$ to be identical to $S$ in the same interval without increasing the makespan of $\hat{S}$ and without violating any precedence constraints.

Using this approach, we can always convert the schedule in $[t_j, r_2]$ in $\hat{S}$ to be identical to $S$ in the same interval without increasing the makespan and without violating any precedence constraints, no matter how many jobs are sharing one processor in the time interval $[t_j, t_{j+1}]$. $\square$

**3.2. Outtrees and arbitrary number of processors.** We first state, without proof, that Algorithm D is an optimal offline algorithm for $P \mid pmtn, outtree_i$ *released at* $r_i \mid C_{\max}$. The proof is similar to Lemma 3.2. We then prove that Algorithm C is an optimal online algorithm for the same case.

THEOREM 3.5. *Algorithm* D *is an optimal offline algorithm for* $P \mid pmtn, outtree_i$ *released at* $r_i \mid C_{\max}$.

THEOREM 3.6. *Algorithm* C *is an optimal online algorithm for* $P \mid pmtn, outtree_i$ *released at* $r_i \mid C_{\max}$.

*Proof.* Let $S$ be the schedule produced by Algorithm C for an instance of $P \mid pmtn, outtree_i$ *released at* $r_i \mid C_{\max}$ and let $\hat{S}$ be the schedule produced by Algorithm D for the same instance. We will show by induction on the number of release times, $i$, that the idle processor time in $S$ is less than or equal to that of $\hat{S}$ at each time instant $t$. Thus, $S$ is an optimal schedule as well.

The basis case, $i = 1$, is obvious, since $S$ and $\hat{S}$ are identical schedules. Assuming that the hypothesis is true for all $i \leq k - 1$, we wish to show that the hypothesis is true for $i = k$. Let $t_1 < t_2 < \cdots < t_l$ be the time instants where event 1 or event 2 occurs in $\hat{S}$. Let $r_1$ and $r_2$ denote the first and second release times, respectively.

There are two cases to consider.

*Case* I. $r_2 = t_j$ for some $1 \leq j \leq l$.

From time $r_1$ until $r_2$, $S$ is identical to $\hat{S}$. At time $r_2$, the remaining portions of the unfinished tasks in both schedules are identical. There are only $k-1$ releases from $r_2$ onward. By the induction hypothesis, the idle processor time in $S$ is less than or equal to that of $\hat{S}$ at each time instant $t$ after $r_2$.

*Case* II. $r_2$ is in the time interval $[t_j, t_{j+1}]$ for some $1 \leq j < l$.

There are two cases to consider.

*Case* II(a). Every task executing in the time interval $[t_j, t_{j+1}]$ is executing on a full processor.

The proof of this case is identical to that of Case I.

*Case* II(b). Some tasks are sharing processor(s) in the time interval $[t_j, t_{j+1}]$.

Let $x$ tasks be sharing $y$ processor(s) $(x > y)$ in the interval $[t_j, t_{j+1}]$. Since tasks are sharing processors, the number of available tasks at time $t_j$ must be greater than the number of processors. Let $i_1, i_2, \ldots, i_x$ be the tasks sharing the $y$ processor(s) and let $j_1, j_2, \ldots, j_z$ be the tasks executing on a full processor. We have $m = y + z$. It is easy to see that from time $r_1$ until $t_j$, $S$ and $\hat{S}$ are identical schedules, but from $t_j$ until $r_2$, $S$ and $\hat{S}$ may not be the same.

The second release time, $r_2$, divides each task $i_1, i_2, \ldots, i_x$ in $S$ into two parts: those that were executed before $r_2$ and those that were executed after $r_2$. Let the level of a task $j$ at time $r_2$ be denoted by $level(j)$. Without loss of generality, assume that $level(j_1) \geq level(j_2) \geq \cdots \geq level(j_z)$. Then we have $level(j_z) \geq level(i_l)$ for each $1 \leq l \leq x$, but the tasks $i_1, i_2, \ldots, i_x$ may have different levels. Without loss of generality, assume that $level(i_1) \geq level(i_2) \geq \cdots \geq level(i_x)$. Then we have $level(j_1) \geq level(j_2) \geq \cdots \geq level(j_z) > level(i_1) \geq level(i_2) \geq \cdots \geq level(i_x)$. Consider now the schedule $\hat{S}$. We have $level(j_1) \geq level(j_2) \geq \cdots \geq level(j_z) > level(i_1) = level(i_2) = \cdots = level(i_x)$.

Consider the schedule $S'$ obtained from $S$ by scheduling the remaining portions of the unfinished tasks at $r_2$ by Algorithm D. Since there are only $k-1$ release times (including $r_2$), by the induction hypothesis, the idle processor time in $S$ is less than or equal to that of $S'$ at each time instant $t$ after $r_2$. Thus, if we can show that the idle processor time in $S'$ is less than or equal to that of $\hat{S}$ at each time instant $t$ after $r_2$, then the theorem is proved. We will prove this assertion by contradiction.

Let $t^*$ be the first time instant where $S'$ has more idle processor time than $\hat{S}$. Clearly, $t^* \geq r_2$. Let $[t^*, t^* + \epsilon]$ be the time interval where $S'$ has more idle processor time than $\hat{S}$ for some small positive number $\epsilon$. Clearly, $S'$ must have some idle processors in the interval $[t^*, t^* + \epsilon]$. Let $i^*$ be the earliest executed task that is executed in the time interval $[r_1, t^* + \epsilon]$ in $\hat{S}$ but not in $S'$. We claim that $i^*$ can also be executed in the interval $[t^*, t^* + \epsilon]$ in $S'$, contradicting the definition of $i^*$.

If $i^*$ cannot be executed in the interval $[t^*, t^* + \epsilon]$ in $S'$, then it must have a predecessor executed in the interval. Let $i^*$ be a task in $outtree_l$, released at time $r_l$. We consider two cases.

*Case* (i). Task $i^*$ is not the successor of any of the task $i_h$, $1 \leq h \leq x$.

Since $i^*$ is not the successor of any of the task $i_h$, $1 \leq h \leq x$, the remaining portions of the predecessors of $i^*$ after $r_2$ in $S'$ must be identical to those in $\hat{S}$. Since $\hat{S}$ schedules $i^*$ by $t^* + \epsilon$ while $S'$ did not, there must be a time interval $[t', t' + \delta]$ such that either (1) in $S'$ the processors are all busy in the interval but none of the predecessors of $i^*$ are executing in the interval; or (2) a predecessor of $i^*$ is assigned fewer processors in the interval in $S'$ than in $\hat{S}$. In both cases there must be more than $m$ tasks ready for execution in the interval $[t', t' + \delta]$ in $S'$. This means that the processors are all busy from $t'$ until $t^* + \epsilon$, contradicting our assumption that $S'$ has some idle processors in the interval $[t^*, t^* + \epsilon]$.

*Case* (ii). Task $i^*$ is the successor of the task $i_h$ for some $1 \leq h \leq x$.

In this case, the remaining portions of the predecessors of $i^*$ after $r_2$ in $S'$ may not be the same as those in $\hat{S}$. If there is a time interval $[t', t' + \delta]$ such that either (1) in $S'$ the processors are all busy in the interval but none of the predecessors of $i^*$ are executing in the interval, or (2) a predecessor of $i^*$ is assigned less processor in the interval in $S'$ than in $\hat{S}$, then we can resort to the same argument as in Case (i). Thus, we may assume that at each time instant after $r_l$, the predecessors of $i^*$ are assigned an equal or greater number of processors in $S'$ than in $\hat{S}$. In this case the only reason that $i^*$ is executed in $\hat{S}$ but not in $S'$ is that the remaining portion of task $i_h$ after $r_2$ is larger in $S'$ than in $\hat{S}$. Let $i_h$ be finished at time $\bar{t}$ in $S'$. It is clear that every one of the tasks $i_1, i_2, \ldots, i_x$ must also be finished at $\bar{t}$. Furthermore, the tasks $j_1, j_2, \ldots, j_z$ are either finished at $\bar{t}$ or still active at $\bar{t}$, since they have higher levels than $i_h$. Thus, there are more active tasks than the number of processors. But this means that the processors are all busy from $\bar{t}$ until $t^* + \epsilon$, contradicting our assumption that $S'$ has some idle processors in the interval $[t^*, t^* + \epsilon]$. $\square$

## 4. Conclusion.

In this article we have given optimal online algorithms for the following problems:

(1) $P2 \mid p_j = 1, prec_i \textit{ released at } r_i \mid C_{\max}$.
(2) $P \mid p_j = 1, outtree_i \textit{ released at } r_i \mid C_{\max}$.
(3) $P2 \mid pmtn, prec_i \textit{ released at } r_i \mid C_{\max}$.
(4) $P \mid pmtn, outtree_i \textit{ released at } r_i \mid C_{\max}$.

We also show that it is impossible to have optimal online algorithms for the following problems:

(1) $P3 \mid p_j = 1, intree_i \textit{ released at } r_i \mid C_{\max}$.
(2) $P2 \mid p_j = p, chains_i \textit{ released at } r_i \mid C_{\max}$.
(3) $P3 \mid pmtn, p_j = 1, intree_i \textit{ released at } r_i \mid C_{\max}$.

Instead of the makespan objective, one wonders whether there are optimal online algorithms for the mean flow time objective. In this regard, it can be shown that Algorithm A is an optimal online algorithm for $P2 \mid p_j = 1, prec_i \textit{ released at } r_i \mid \sum C_j$, while Algorithm B is an optimal online algorithm for $P \mid p_j = 1, outtree_i \textit{ released at } r_i \mid \sum C_j$. The proof in Theorem 2.1 also shows that it is impossible to have optimal online algorithms for $P3 \mid p_j = 1, intree_i \textit{ released at } r_i \mid \sum C_j$ and $P3 \mid pmtn, p_j = 1, intree_i \textit{ released at } r_i \mid \sum C_j$. Relatively little is known about preemptive scheduling. For example, is it possible to have an optimal online algorithm for $P2 \mid pmtn, p_j = 1, prec_i \textit{ released at } r_i \mid \sum C_j$? Recently, Coffman, Sethuraman, and Timkovsky [2] gave an algorithm that simultaneously minimizes the makespan and the mean flow time for $P2 \mid pmtn, p_j = 1, prec_i \textit{ released at } r_i \mid \sum C_j$. Is it possible to adapt their algorithm to yield an optimal online algorithm for this case?

## REFERENCES

[1] E. G. Coffman, Jr., and R. L. Graham, *Optimal scheduling for two-processor systems*, Acta Inform., 1 (1972), pp. 200–213.
[2] E. G. Coffman, Jr., J. Sethuraman, and V. G. Timkovsky, *Ideal preemptive schedules on two processors*, Acta Inform., 39 (2003), pp. 597–612.
[3] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, W. H. Freeman and Company, San Francisco, 1979.

[4] R. L. Graham, E. L. Lawler, J. K. Lenstra, and A. H. G. Rinnooy Kan, *Optimization and approximation in deterministic sequencing and scheduling: A survey*, Ann. Discrete Math., 5 (1979), pp. 287–326.
[5] K. S. Hong and J. Y.-T. Leung, *On-line scheduling of real-time tasks*, IEEE Trans. Comput., 41 (1992), pp. 1326–1331.
[6] T. C. Hu, *Parallel sequencing and assembly line problems*, Oper. Res., 9 (1961), pp. 841–848.
[7] R. McNaughton, *Scheduling with deadlines and loss functions*, Management Sci., 6 (1959), pp. 1–12.
[8] R. R. Muntz and E. G. Coffman, Jr., *Optimal preemptive scheduling on two-processor systems*, IEEE Trans. Comput., 18 (1969), pp. 1014–1020.
[9] R. R. Muntz and E. G. Coffman, Jr., *Preemptive scheduling of real-time tasks on multiprocessor systems*, J. ACM, 17 (1970), pp. 324–338.

SIAM J. Comput. © 2005 Society for Industrial and Applied Mathematics
Vol. 34, No. 3, pp. 763–773

# PRIMAL-DUAL MEETS LOCAL SEARCH: APPROXIMATING MSTs WITH NONUNIFORM DEGREE BOUNDS[*]

J. KÖNEMANN[†] AND R. RAVI[‡]

**Abstract.** We present a new bicriteria approximation algorithm for the degree-bounded minimum-cost spanning tree (MST) problem: Given an undirected graph with nonnegative edge weights and a degree bound $B$, find a spanning tree of maximum node-degree $B$ and minimum total edge-cost. Our algorithm outputs a tree of maximum degree at most a constant times $B$ and total edge-cost at most a constant times that of a minimum-cost degree-$B$-bounded spanning tree.

While our new algorithm is based on ideas from Lagrangian relaxation, as is our previous work [*SIAM J. Comput.*, 31 (2002), pp. 1783–1793], it does not rely on computing a solution to a linear program. Instead, it uses a repeated application of Kruskal's MST algorithm interleaved with a combinatorial update of approximate Lagrangian node-multipliers maintained by the algorithm. These updates cause subsequent repetitions of the spanning tree algorithm to run for longer and longer times, leading to overall progress and a proof of the performance guarantee. A second useful feature of our algorithm is that it can handle nonuniform degree bounds on the nodes: Given distinct bounds $B_v$ for every node $v \in V$, the output tree has degree at most $O(B_v + \log |V|)$ for every $v \in V$. As before, the cost of the tree is at most a constant times that of a minimum-cost tree obeying all degree bounds.

**Key words.** approximation algorithms, network algorithms, bicriteria approximation, spanning trees, degree-bounded spanning trees, Lagrangian relaxation

**AMS subject classifications.** 68W25, 05C05, 05C85, 68R10, 90C29

**DOI.** 10.1137/S0097539702418048

## 1. Introduction.

### 1.1. Formulation.

In this paper, we address a natural nonuniform budget version of the *degree-bounded minimum-cost spanning tree problem* (nBMST). Given an undirected graph $G = (V, E)$, a cost function $c : E \to \mathbb{R}^+$, and positive integers $\{B_v\}_{v \in V}$ all greater than 1, our goal is to find a spanning tree $T$ of minimum total cost such that for all vertices $v \in V$ the degree of $v$ in $T$ is at most $B_v$.

### 1.2. Previous work and our result.

In an $n$-node graph, Ravi et al. [11, 12] showed how to compute a spanning tree $T$ of degree $O(B_v \log n)$ for all $v \in V$ and total cost at most $O(\log n)\,\texttt{opt}$, where $\texttt{opt}$ is the minimum cost of any tree in which the degree of node $v$ is bounded by $B_v$ for all $v \in V$. The authors generalized their ideas to Steiner trees, generalized Steiner forests, and the node-weighted case. Subsequently, Könemann and Ravi [8, 9] improved the results on the (edge-weighted) spanning tree version of the problem with uniform degree bounds (i.e., $B_v = B$ for all $v$). The main theorem from [9] is as follows.

Theorem 1.1 (see [9]). *There is an approximation algorithm that, given a graph $G = (V, E)$, a nonnegative cost function $c : E \to \mathbb{R}^+$, a constant $B \geq 2$, and a parameter $\omega > 1$, computes a spanning tree $T$ such that*

1. $\Delta(T) \leq \frac{\omega}{\omega-1} \cdot bB + \log_b n$, *and*

[*]Received by the editors November 15, 2002; accepted for publication (in revised form) September 14, 2004; published electronically April 19, 2005. This material is based upon work supported by the National Science Foundation under grant 0105548.
http://www.siam.org/journals/sicomp/34-3/41804.html

[†]Department of Combinatorics and Optimization, University of Waterloo, 200 University Avenue West, Waterloo, ON N2L 3G1, Canada (jochen@math.uwaterloo.ca).

[‡]GSIA, Carnegie Mellon University, Pittsburgh, PA 15213 (ravi@cmu.edu).

2. $c(T) < \omega \cdot \texttt{opt}$,

*where $b > 1$ is an arbitrary constant.*

The contributions of this paper are twofold: First, we present improved approximation algorithms for the nBMST problem in the presence of nonuniform degree bounds. Second, our algorithm is *direct* in the sense that we do not solve linear programs. The algorithm in [9] uses Lagrangian relaxation and thus needs to solve a linear program. The analysis in [9] relies crucially on the fact that we compute an exact solution to this linear program.

On the other hand, our new algorithm integrates elements from the primal-dual method for approximation algorithms for network design problems with local search methods for minimum-degree network problems [5, 13]. The algorithm goes through a series of spanning trees and improves the maximum deviation of any vertex degree from its respective degree bound progressively. A practical consequence of this is that we can terminate the algorithm at any point in time and still obtain a spanning tree of the input graph (whose node-degrees, of course, may not meet the worst-case guarantees we prove).

THEOREM 1.2. *There is a primal-dual approximation algorithm that, given a graph $G = (V, E)$, a nonnegative cost function $c : E \to \mathbb{R}^+$, integers $B_v > 1$ for all $v \in V$, and a parameter $\omega > 1$, computes a tree $T$ such that*

1. $\deg_T(v) \le \frac{\omega}{\omega-1} \cdot b \cdot B_v + 2 \cdot \log_b n$ *for all $v \in V$, and*
2. $c(T) < \omega \cdot \texttt{opt}$,

*where $b > 1$ is an arbitrary constant. The running time is $O(|E||V|^5 \log |V|)$.*

When $B = B_v$ for all $v \in V$, we can replace the additive term $2 \log_b n$ in the degree guarantee of Theorem 1.2 by $\log_b n$, matching Theorem 1.1 constructively.

The paper is organized as follows. First, we review the primal-dual interpretation of the well-known algorithm for MST by Kruskal [10]. Subsequently, we show how to use this algorithm for the nBMST problem and present an analysis of the performance guarantee and the running time of our method.

## 2. A primal-dual algorithm to compute MSTs.

In this section we review Kruskal's MST algorithm. More specifically, we discuss a primal-dual interpretation of this method that follows from [2]. We start by giving a linear programming formulation of the convex hull of incidence vectors of spanning trees.

### 2.1. The spanning tree polyhedron.

In the following, we formulate the MST problem as an integer program where we associate a $(0, 1)$-variable $x_e$ with every edge $e \in E$. In a solution $x$, the value of $x_e$ is 1 if $e$ is included in the spanning tree corresponding to $x$ and 0 otherwise. Our formulation relies on a complete formulation of the convex hull of incidence vectors of spanning trees (denoted by $\texttt{SP}_G$) given by Chopra [2].

Chopra's formulation uses the notion of a *feasible partition* of vertex set $V$. A feasible partition of $V$ is a set $\pi = \{V_1, \dots, V_k\}$ such that the $V_i$ are pairwise disjoint subsets of $V$. Moreover, $V = V_1 \cup \dots \cup V_k$ and the induced subgraphs $G[V_i]$ are connected. Let $G_\pi$ denote the (multi-) graph that has one vertex for each $V_i$ and in which edge $(V_i, V_j)$ occurs with multiplicity $|\{(v_i, v_j) : v_i \in V_i, v_j \in V_j\}|$. In other words, $G_\pi$ results from $G$ by contracting each $V_i$ to a single node. Define the *rank* $r(\pi)$ of $\pi$ as the number of nodes of $G_\pi$ and let $\Pi$ be the set of all feasible partitions of $V$. Chopra then shows that

$$\texttt{SP}_G = \left\{ x \in \mathbb{R}^m : \sum_{e \in E(G_\pi)} x_e \ge r(\pi) - 1 \quad \forall \pi \in \Pi \right\}.$$

We now let $\delta(v)$ denote the set of edges $e \in E$ that are incident to node $v$ and obtain an integer programming (IP) formulation for our problem:

$$
\begin{aligned}
\text{(IP)} \qquad & \min \quad \sum_{e\in E} c_e x_e \\
& \text{s.t.} \quad \sum_{e \in E[G_\pi]} x_e \geq r(\pi) - 1 \quad \forall \pi \in \Pi, \\
\text{(2.1)} \qquad & \qquad x(\delta(v)) \leq B_v \quad \forall v \in V, \\
& \qquad x \text{ integer.}
\end{aligned}
$$

The dual of the linear programming relaxation of (IP) is given by

$$
\begin{aligned}
\text{(D)} \qquad & \max \quad \sum_{\pi\in\Pi} (r(\pi) - 1) \cdot y_\pi - \sum_{v \in V} \lambda_v B_v \\
\text{(2.2)} \qquad & \text{s.t.} \quad \sum_{\pi : e \in E[G_\pi]} y_\pi \leq c_e + \lambda_u + \lambda_v \quad \forall e = uv \in E, \\
& \qquad y, \lambda \geq 0.
\end{aligned}
$$

We also let (IP-SP) denote (IP) without constraints of type (2.1). Let the linear programming relaxation be denoted by (LP-SP) and let its dual be (D-SP).

**2.2. A primal-dual interpretation of Kruskal's MST algorithm.** Kruskal's algorithm can be viewed as a continuous process over *time*: We start with an empty tree at time 0 and add edges as we go along. The algorithm terminates at time $t^*$ with a spanning tree of the input graph $G$. In this section we show that Kruskal's method can be interpreted as a primal-dual algorithm (see also [7]). For that reason, at any time $0 \leq t \leq t^*$ we keep a pair $(x_t, y_t)$, where $x_t$ is a partial (possibly infeasible) primal solution for (LP-SP) and $y_t$ is a feasible dual solution for (D-SP). Initially, we let $x_{e,0} = 0$ for all $e \in E$ and $y_{\pi,0} = 0$ for all $\pi \in \Pi$.

Let $E_t$ be the forest corresponding to partial solution $x_t$, i.e., $E_t = \{e \in E : x_{e,t} = 1\}$. We then denote by $\pi_t$ the partition induced by the connected components of $G[E_t]$. At time $t$, the algorithm then increases $y_{\pi_t}$ until a constraint of type (2.2) for edge $e \in E \setminus E_t$ becomes tight. Assume that this happens at time $t' > t$. The dual update is

$$y_{\pi_t, t'} = t' - t.$$

We then include $e$ in our solution, i.e., we set $x_{e,t'} = 1$. If more than one edge becomes tight at time $t'$, we can process these events in any arbitrary order. Thus, note that we can pick any such tight edge first in our solution.

Observe that the above primal-dual algorithm is indeed Kruskal's algorithm: If the algorithm adds an edge $e$ at time $t$, then $e$ is the minimum-cost edge connecting two connected components of $G[E_t]$. In the rest of this paper we use `MST` to refer to Kruskal's minimum-cost spanning tree algorithm.

The proof of the following lemma is folklore. We supply it for the sake of completeness.

LEMMA 2.1. *At time $t^*$, algorithm* `MST` *finishes with a pair $(x_{t^*}, y_{t^*})$ of primal and dual feasible solutions to* (IP-SP) *and* (D-SP)*, respectively, such that*

$$\sum_{e \in E} c_e x_{e,t^*} = \sum_{\pi \in \Pi} (r(\pi) - 1) \cdot y_{\pi, t^*}.$$

*Proof.* Notice that for all edges $e \in E_{t^*}$ we must have $c_e = \sum_{\pi : e \in E[G_\pi]} y_{\pi,t^*}$ and hence, we can express the cost of the final tree as follows:

$$c(E_{t^*}) = \sum_{e \in E_{t^*}} \sum_{\pi : e \in E[G_\pi]} y_{\pi,t^*} = \sum_{\pi \in \Pi} |E_{t^*} \cap E[G_\pi]| \cdot y_{\pi,t^*}.$$

By construction $E_{t^*}$ is a tree and we must have that the set $E_{t^*} \cap E[G_\pi]$ has cardinality exactly $r(\pi) - 1$ for all $\pi \in \Pi$ with $y_{\pi,t^*} > 0$. We obtain that $\sum_{e \in E} c_e x_{e,t^*} = \sum_{\pi \in \Pi} (r(\pi) - 1) \cdot y_{\pi,t^*}$ and this finishes the proof of the lemma. □

**3. Minimum-cost degree-bounded spanning trees.** In this section, we propose a modification of the above algorithm for approximating degree-bounded spanning trees of low total cost (for suitably weakened degree bounds). Our algorithm goes through a sequence of spanning trees $E^0, \dots, E^t$ and associated pairs of primal (infeasible) and dual feasible solutions $x^i$, $(y^i, \lambda^i)$ for $0 \leq i \leq t$. The idea is to reduce the degree of nodes $v \in V$ whose degree is substantially higher than their associated bound $B_v$, as we proceed through this sequence, while keeping the cost of the associated primal solution (tree) bounded with respect to the corresponding dual solution.

To begin, our algorithm first computes a minimum-cost spanning tree using the algorithm MST. This yields a feasible primal solution $x^0$ for (LP-SP) and a feasible dual solution $y^0$ for (D-SP). Notice that $y^0$ also induces a feasible solution for (D) by letting $\lambda^0_v = 0$ for all $v \in V$, while $x^0$ potentially violates constraints of type (2.1).

We introduce the notion of *normalized degree* of a node $v$ in a tree $T$ and denote it by

$$\texttt{ndeg}_T(v) = \max\{0, \deg_T(v) - \beta \cdot B_v\}, \tag{3.1}$$

where $\beta \geq 1$ is a constant to be specified later. Our algorithm successively computes pairs of spanning trees and associated dual solutions to (D), i.e.,

$$(x^1, \{y^1, \lambda^1\}), (x^2, \{y^2, \lambda^2\}), \dots, (x^t, \{y^t, \lambda^t\}).$$

From one such pair to the next, we try to reduce the degree of nodes of high normalized degree. Specifically, our algorithm runs as long as there is a node in the current tree with $\texttt{ndeg}(v) \geq 2 \log_b(n)$ for some constant $b > 1$.

Our algorithm keeps a cost $\widetilde{c}^i_e$ with each edge $e \in E$ and for each iteration $1 \leq i \leq t$. $x^i$ corresponds to an MST $E^i$ for cost function $\widetilde{c}^i$, and $y^i$ is the associated dual packing. Throughout the algorithm we maintain that

$$c_{uv} \leq \widetilde{c}^i_{uv} \leq c_{uv} + \lambda^i_u + \lambda^i_v \tag{3.2}$$

for all $uv \in E$. Hence, $(y^i, \lambda^i)$ is a feasible solution for (D).

Let $\Delta^i$ be the maximum normalized degree of any node in the tree $E^i$. The central piece of our algorithm is a *recompute step*, where we raise the $\lambda$ values of a carefully chosen set $S_d$ of nodes with high normalized degree. This introduces slack in many of the constraints of type (2.2).

We now increase the $\widetilde{c}$-cost of edges that are incident to nodes in $S_d$ (while maintaining (3.2)) and rerun MST on $G$ using the new edge-costs. Our hope is that the increased $\widetilde{c}$-cost of edges incident to nodes of high normalized degree leads MST to use edges that are incident to nodes of lower normalized degree in their place. We are able to show that the number of recompute steps is polynomial by arguing that we make substantial progress in the normalized degree sequence of all nodes.

As mentioned, each recompute step takes a pair of primal infeasible and dual feasible solutions $(x^i, (y^i, \lambda^i))$ and computes a new pair of primal (infeasible) and dual feasible solutions $(x^{i+1}, (y^{i+1}, \lambda^{i+1}))$. In the following we use $\texttt{ndeg}^i(v)$ as shorthand for $\texttt{ndeg}_{E^i}(v)$. We then adapt the notation from [4, 5] and let

$$S^i_d = \{v \in V : \texttt{ndeg}^i(v) \geq d\}$$

be the set of all nodes whose normalized degree is at least $d$ in the $i$th solution.

**Algorithm 1** The algorithm for the nBMST problem attempts to reduce the maximum normalized degree of any node in a given spanning tree.

```
1: Given: primal feasible solution $x^0$ to (LP-SP) and dual feasible solution $y^0$ to (D-SP)
2: $\lambda^0_v \leftarrow 0 \forall v \in V; \widetilde{c}^0_e \leftarrow c_e, \forall e \in E$
3: $i \leftarrow 0$
4: while $\Delta^i > 2\log_b(n)$ do
5:    Choose $d^i$ in $\{\Delta^i - 2\log_b(n)+1, \ldots, \Delta^i\}$ s.t. $\sum_{v \in S^i_{d^i-1}} B_v \leq b \cdot \sum_{v \in S^i_{d^i}} B_v$
6:    Choose $\epsilon^i > 0$
7:    $\lambda^{i+1}_v \leftarrow \lambda^i_v + \epsilon^i \forall v \in S^i_{d^i-1}$ and $\lambda^{i+1}_v \leftarrow \lambda^i_v$ otherwise
8:    $\widetilde{c}^{i+1}(e) \leftarrow \widetilde{c}^i(e) + \epsilon^i$ if either $e \in E^i$ and $e \cap S^i_{d^i} \neq \emptyset$ or $e \notin E^i$ and $e \cap S^i_{d^i-1} \neq \emptyset$
9:    $(x^{i+1}, y^{i+1}) \leftarrow \texttt{MST}(G, \widetilde{c}^{i+1})$
10:   $i \leftarrow i+1$
11: end while
```

A detailed description of the procedure is given in Algorithm 1. In step 5 of this algorithm, we choose a suitable set of nodes whose $\lambda$-values we increase. A simple argument in [4] can be extended to guarantee the feasibility of the choice in step 5 of the algorithm.

LEMMA 3.1. *There is a $d^i \in \{\Delta^i - 2\log_b(n)+1, \ldots, \Delta^i\}$ such that*

$$\sum_{v \in S^i_{d^i-1}} B_v \leq b \cdot \sum_{v \in S^i_{d^i}} B_v$$

*for a given constant $b > 1$.*

*Proof.* Suppose for a contradiction that for all $d^i \in \{\Delta^i - 2\log_b(n)+1, \ldots, \Delta^i\}$, we have

$$\sum_{v \in S^i_{d^i-1}} B_v > b \cdot \sum_{v \in S^i_{d^i}} B_v.$$

Note that since we may assume $B_v \leq (n-1)$ for all vertices, we must have $\sum_{v \in V} B_v \leq n(n-1)$. However, since $\sum_{v \in S^i_{\Delta^i}} B_v \geq 1$, we have in this case that

$$\sum_{v \in S^i_{\Delta^i - 2\log_b(n)}} B_v \geq b^{2\log_b(n)} = n^2,$$

a contradiction. □

When $B_v = B$ for all $v \in V$, the $2\log_b n$ term in the above lemma can be improved to $\log n$ using the previous arguments in [4]. This in turn leads to the slight improvement of our results claimed right after the statement of Theorem 1.2.

Step 6 of Algorithm 1 hides the details of choosing an appropriate $\epsilon^i$ by which edges in the current tree that are incident to nodes of normalized degree at least $d^i$ are lengthened. Our choice of $\epsilon^i$ and the following update of the $\widetilde{c}$ costs of the edges in $G$ will ensure that Kruskal's algorithm computes a new tree in which at least one edge $e^i \in E^i$ that is incident to a node of $S^i_{d^i}$ is replaced by an edge $\overline{e}^i \notin E^i$ that is incident to nodes of low normalized degree.

In fact, we show that a careful choice of $\epsilon^i$ ensures that $\overline{e}^i$ is incident to nodes of normalized degree at most $d^i-2$, while $e^i$ is incident to at least one node of normalized degree $d^i$ or higher. The main idea here is to increase $\lambda_v$ for nodes $v \in S^i_{d^i-1}$ by $\epsilon^i$ and increase the $\widetilde{c}$-cost of nontree edges that are incident to nodes of normalized degree at least $d^i - 1$ by $\epsilon^i$ as well. In other words, the cost of nontree edges incident to nodes of normalized degree at least $d^i - 1$ increases by the same amount as the cost of tree edges incident to nodes of normalized degree at least $d^i$. This way, we enforce that the edge we swap in touches nodes of normalized degree at most $d^i - 2$. Once we accomplish this, by adapting a potential function argument from [4] we can put a polynomial upper bound on the number of such iterations (see section 3.3).

Lemma 3.1 plays a key role in the later analysis of the performance guarantee achieved by Algorithm 1. Notice that step 7 of the algorithm increases the node multipliers of all nodes in $S^i_{d^i-1}$. On the other hand, in step 8, we increase only the $\widetilde{c}$-cost of those tree edges that are incident to nodes in $S^i_{d^i}$. Roughly, Lemma 3.1 provides us with a bound on the number of tree edges that are incident to nodes of normalized degree exactly $d^i - 1$.

We now describe how to choose $\epsilon^i$ so that the above conditions are satisfied.

## 3.1. Choosing $\epsilon^i$.

In this section we elaborate on the choice of $\epsilon^i$ in step 6 of Algorithm 1. In step 8, we increase $\widetilde{c}_{uv}$ by $\epsilon^i$ for all tree edges $uv$ that are incident to nodes of degree at least $d^i$ and for all nontree edges that are incident to nodes of degree at least $d^i-1$. We want to choose $\epsilon^i$ such that the subsequent update of $\widetilde{c}^i$ and the following run of `MST` yield a new tree $E^{i+1}$ that differs from $E^i$ by a single edge swap: $E^{i+1} = E^i \setminus \{e^i\} \cup \overline{e}^i$. Here, the edge $e^i \in E^i$ is a tree edge that is incident to a node from $S^i_{d^i}$. On the other hand we want $\overline{e}^i \in E \setminus E^i$ to be a nontree edge that is not incident to any node from $S^i_{d^i-1}$.

As indicated in the previous section, we want $E^{i+1}$ to be an MST of $G$ for cost function $\widetilde{c}^{i+1}$. In order to achieve this, $e^i$ must be on the unique cycle in $E^i \cup \{\overline{e}^i\}$ and we also must have

$$\widetilde{c}^{i+1}(e^i) = \widetilde{c}^i(e^i) + \epsilon^i = \widetilde{c}^{i+1}(\overline{e}^i).$$

In other words, the update of $\lambda_v$ for $v \in S^i_{d^i-1}$ creates one more beneficial swap.

We let $\mathcal{K}^i$ be the set of connected components of the forest $E^i \setminus S^i_{d^i}$, i.e., the forest that results from removing nodes of normalized degree at least $d^i$ from $E^i$. We say that an edge $e = uv \in E$ is a *cross-edge* if

1. $e$ is a nontree edge, i.e., $e \in E \setminus E^i$,
2. $u \in K_1, v \in K_2$ for $K_1, K_2 \in \mathcal{K}^i$, and $K_1 \neq K_2$, and
3. $\{u, v\} \cap S^i_{d^i-1} = \emptyset$.

We denote the set of cross-edges in iteration $i$ by $\mathcal{C}^i$. Observe that if $C^i = \emptyset$, then the set $S_{d^i-1}$ provides a witness to the infeasibility of the degree bounds imposed on the nodes in this set.

It is now clear that $E^i + e$ contains a unique cycle $C_e^i$ for each cross-edge $e \in \mathcal{C}^i$. Furthermore, there must be at least one vertex $v$ on $C_e^i$ that has normalized degree at least $d^i$.

For each cross-edge $e \in \mathcal{C}^i$, we now let

$$\epsilon_e^i = \min_{e' \in C_e^i, e' \cap S_{d^i}^i \neq \emptyset} \left(\widetilde{c}^i(e) - \widetilde{c}^i(e')\right).$$

Note that it follows from the fact that $E^i$ is an MST for cost function $\widetilde{c}^i$ that $\epsilon_e^i \geq 0$ for all $e \in \mathcal{C}^i$. Finally, we let $\epsilon^i = \min_{e \in \mathcal{C}^i} \epsilon_e^i$.

In the following, we let $\langle e^i, \overline{e}^i \rangle$ be the *witness pair* for $\epsilon^i$. In other words, let $\langle f, \overline{f} \rangle$ be a pair of edges where $\overline{f} \in E \setminus E^i$ is a nontree edge and $f \in C_{\overline{f}}^i$ is a tree edge that is incident to a node from $S_{d^i}^i$ and that lies on the unique cycle in $E^i + \overline{f}$. Then, we must have that

$$\widetilde{c}^i(f) + \epsilon^i \leq \widetilde{c}^i(\overline{f}),$$

and equality holds for $f = e^i$ and $\overline{f} = \overline{e}^i$. Notice that $\overline{e}^i$ is incident to nodes of normalized degree at most $d^i - 2$ by the definition of cross-edges.

An important observation is that $\epsilon^i$ can be 0. Such a step can be viewed as a *local-improvement* step along the lines of [5]. We do not modify the dual solution but decrease the normalized degree of a node of high normalized degree.

**3.2. Analysis: Performance guarantee.** Assume that Algorithm 1 terminates after iteration $t^*$. Recall that Theorem 1.2 requires us to show

$$c(E^{t^*}) \leq \omega \cdot \mathtt{opt}\,. \tag{3.3}$$

In this section we prove

$$c(E^i) = \sum_{e \in E^i} c_e \leq \omega \sum_{\pi \in \Pi} (r(\pi) - 1) \cdot y_\pi^i - \omega \cdot \sum_{v \in V} B_v \cdot \lambda_v^i \tag{3.4}$$

for all $1 \leq i \leq t^*$. Observe that $(y^i, \lambda^i)$ is a feasible solution for (D) and that the right-hand side of (3.4) is $\omega$ times the dual objective function induced by $(y^i, \lambda^i)$. Inequality (3.4) for $i = t^*$ together with weak duality implies (3.3). This line of proof extends that used in the analysis of primal-dual algorithms for minimum-cost networks developed in [1, 6].

In order to facilitate the proof of (3.4) for all $1 \leq i \leq t^*$, we maintain the following invariant inductively for all $0 \leq i \leq t^*$:

$$\omega \cdot \sum_{v \in V} B_v \lambda_v^i \leq (\omega - 1) \cdot \sum_{\pi \in \Pi} (r(\pi) - 1) \cdot y_\pi^i. \tag{Inv}$$

Recall that we assumed $\omega > 1$ in Theorem 1.2.

We now use the above invariant to provide a short proof of (3.4). Later in section 3.2.2, we prove (Inv) itself.

**3.2.1. Bounding the cost of $\boldsymbol{E^i}$.** Recall that we have $c_e \leq \widetilde{c}_e^i$ for all $e \in E$ and for all $1 \leq i \leq t$. As MST finishes we obtain from Lemma 2.1 that

$$c(E^{i+1}) \leq \widetilde{c}^{i+1}(E^{i+1}) = \sum_{e \in E^{i+1}} \widetilde{c}_e^{i+1} = \sum_{\pi \in \Pi} (r(\pi) - 1) \cdot y_\pi^{i+1}.$$

Adding (Inv) for iteration $(i+1)$ to the last inequality, we get

$$c(E^{i+1}) \leq \omega \cdot \left( \sum_{\pi \in \Pi} (r(\pi) - 1) \cdot y_\pi^{i+1} - \sum_{v \in V} B_v \lambda_v^{i+1} \right).$$

This finishes the proof of the performance guarantee claimed in Theorem 1.2. It remains to prove the invariant (Inv) for all $0 \leq i \leq t^*$.

**3.2.2. Proof of invariant (Inv).** We prove the validity of (Inv) for all $1 \leq i \leq t^*$ by induction over $i$. First, notice that (Inv) holds for $i = 0$ since $\lambda_v^0 = 0$ for all $v \in V$. To see the induction step of (Inv) we use the following lemma that ultimately yields (3.4). Recall the definition of normalized degree in (3.1) and the role of the parameter $\beta > 0$ in it.

LEMMA 3.2. *Let $b > 1$ be the constant chosen in Theorem 1.2. We must have*

$$\sum_{\pi \in \Pi} (r(\pi) - 1) \cdot y_\pi^{i+1} \geq \sum_{\pi \in \Pi} (r(\pi) - 1) \cdot y_\pi^i + \frac{\epsilon^i \beta}{b} \cdot \sum_{v \in S^i_{d^i - 1}} B_v$$

*for all $0 \leq i \leq t^*$.*

The lemma quantifies the increase in the dual objective function value as our algorithm moves from $y^i$ to $y^{i+1}$. Intuitively, the lemma shows that the increase in the dual objective function value is proportional to the total slack created by lengthening tree edges of $E^i$ that are incident to nodes of normalized degree at least $d^i - 1$.

Before presenting the proof of Lemma 3.2 we will use it to prove (Inv). Observe that the left-hand side of (Inv) increases by $\omega \epsilon^i \sum_{v \in S^i_{d^i-1}} B_v$ as a consequence of increasing the $\lambda$-values of nodes in $S^i_{d^i-1}$. Lemma 3.2 implies that the right-hand side of (Inv) increases by at least

$$(\omega - 1) \cdot \frac{\beta \epsilon^i}{b} \cdot \sum_{v \in S^i_{d^i-1}} B_v.$$

Choosing

$$\beta \geq b \cdot \frac{\omega}{\omega - 1} \tag{3.5}$$

completes the proof of invariant (Inv). Notice that this choice of $\beta$ together with the definition of normalized degree in (3.1) implies the degree-bound stated in Theorem 1.2.

We now prove Lemma 3.2.

*Proof of Lemma* 3.2. Let $E^i = \{e_1^i, \ldots, e_{n-1}^i\}$ and let $t_j^i$ be the time at which MST included edge $e_j^i$. W.l.o.g., assume that $t_1^i \leq \cdots \leq t_{n-1}^i$. From the description of MST we can rewrite

(3.6)
$$\sum_{\pi \in \Pi} (r(\pi) - 1) \cdot y_\pi^i = \sum_{j=1}^{n-1} (t_j^i - t_{j-1}^i) \cdot (n - j) = \sum_{j=1}^{n-1} t_j^i \left((n - j + 1) - (n - j)\right) = \sum_{j=1}^{n-1} t_j^i,$$

where we define $t_0^i = 0$.

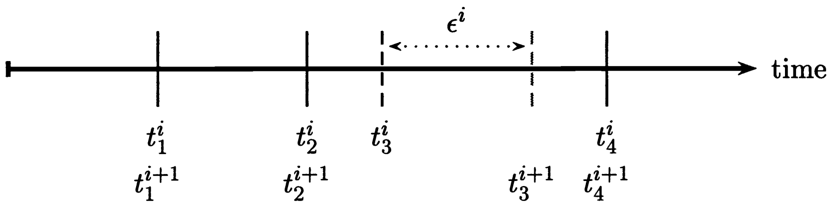


FIG. 3.1. *The figure shows two runs of* MST *in two consecutive iterations, $i$ and $i+1$. The horizontal line is the time axes while the vertical lines denote the times at which different edges become tight and are included. In the $i$th run edge $e_l$ becomes tight at time $t_l^i$. In this example, edge $e_3$ is the only edge incident to a vertex of normalized degree at least $d^i$. Its length increases and the time at which it becomes tight during* MST*'s execution is postponed by $\epsilon^i$ time units.*

Figure 3.1 illustrates the effect of iteration $i$. Observe that, by our choice of $\epsilon^i$, $E^i$ is an MST even for cost function $\widetilde{c}^{i+1}$. However, recall that there is a way to break ties such that step 9 in iteration $i$ correctly outputs $E^{i+1} = E^i \setminus \{e^i\} \cup \{\overline{e}^i\}$. This observation together with (3.6) enables us to quantify the change in dual in iteration $i$:

$$\sum_{\pi\in\Pi}(r(\pi)-1)\cdot\left(y_\pi^{i+1}-y_\pi^i\right) = \sum_{j=1}^{n-1}\left(t_j^{i+1}-t_j^i\right). \tag{3.7}$$

In iteration $i$, we increase the $\widetilde{c}$-cost of all edges $e \in E^i$ that are incident to nodes of normalized degree at least $d^i$ by $\epsilon^i$ while the $\widetilde{c}$-costs of all other tree edges remain unchanged. It is not hard to see that the time an edge becomes tight equals its $\widetilde{c}$-cost. In other words, all edges in $E^i$ that are incident to nodes of normalized degree at least $d^i$ become tight $\epsilon^i$ time units later. Together with (3.7) we obtain

$$\sum_{\pi\in\Pi}(r(\pi)-1)\cdot\left(y_\pi^{i+1}-y_\pi^i\right) = \epsilon^i\cdot\left|E\left(S_{d^i}^i\right)\cap E^i\right|, \tag{3.8}$$

where $E\left(S_{d^i}^i\right)$ denotes the set of edges in $E$ that are incident to nodes from $S_{d^i}^i$. (Note that we include in $E(S)$ edges with both endpoints in $S$.)

Recall the definition of normalized degree in (3.1). Notice that it follows from the termination condition in step 4 of Algorithm 1 that $\Delta^i > 2\log_b(n)$ and hence $d^i > 0$. Therefore, the real degree of any node $v \in S_{d^i}^i$ must be at least

$$\beta\cdot B_v + d_i \geq \beta\cdot B_v + 1.$$

Finally, notice that it follows from the fact that $E^i$ is a tree that there are at most $|S_{d^i}^i|-1$ edges in $E\left(S_{d^i}^i\right)$ that are incident to two nodes from $S_{d^i}^i$. We can use these observations to lower-bound the right-hand side of (3.8):

$$\sum_{\pi\in\Pi}(r(\pi)-1)\cdot\left(y_\pi^{i+1}-y_\pi^i\right) \geq \epsilon^i\cdot\left(\left(\sum_{v\in S_{d^i}^i}\beta\cdot B_v+1\right)-\left(|S_{d^i}^i|-1\right)\right) \geq \epsilon^i\beta\cdot\sum_{v\in S_{d^i}^i}B_v.$$

An application of Lemma 3.1 yields the lemma. ☐

**3.3. Analysis: Running time.** In this section, we show that Algorithm 1 terminates in polynomial time. We accomplish this by showing that there will be only a polynomial number of iterations of the main loop in Algorithm 1.

LEMMA 3.3. *Algorithm* 1 *terminates after* $O(n^4)$ *iterations.*

*Proof.* Following [4], we define the potential of spanning tree $E^i$ as

$$\Phi_i = \sum_{v \in V} 3^{\mathtt{ndeg}_i(v)},$$

where $\mathtt{ndeg}_i(v)$ denotes again the normalized degree of node $v$ in the tree $E^i$.

Notice that an iteration of Algorithm 1 swaps out a single edge $e^i$ that is incident to at least one node of normalized degree at least $d^i$. On the other hand, we swap in one edge $\overline{e}^i$ that is incident to two nodes of normalized degree at most $d^i - 2$. Hence, the reduction in the potential is at least

$$(3^{d^i} + 2 \cdot 3^{d^i-2}) - 3 \cdot 3^{d^i-1} \geq 2 \cdot 3^{d^i-2}.$$

Using the range of $d^i$, we can lower-bound the right-hand side of the last inequality by

$$2 \cdot 3^{\Delta^i - 2\log_b(n) - 2} = \Omega\left(\frac{3^{\Delta^i}}{n^2}\right).$$

The initial potential $\Phi_i$ is at most $n \cdot 3^{\Delta^i}$ and the decrease in the potential $\Phi_i$ in iteration $i$ is at least $\Omega\left(\frac{\Phi_i}{n^3}\right)$.

In other words, in $O(n^3)$ iterations, we reduce $\Phi$ by a constant factor. Since the initial potential is at most $n \cdot 3^n$ and the final potential is at least $n \cdot 3^2$, we obtain that Algorithm 1 terminates in $O(\log(n \cdot 3^n/(n \cdot 3^2))) = O(n)$ phases of $O(n^3)$ iterations each. Hence, the algorithm runs for $O(n^4)$ iterations total. Observing that each iteration can be implemented in time $O(m \log n)$ using standard priority-queue techniques (e.g., see [3]), we see that the whole algorithm runs in time $O(mn^5 \log n)$. ☐

**Acknowledgment.** R. Ravi thanks IBM Almaden for hosting his stay during the writing of this paper.

## REFERENCES

[1] A. AGRAWAL, P. KLEIN, AND R. RAVI, *When trees collide: An approximation algorithm for the generalized Steiner problem in networks*, SIAM J. Comput., 24 (1995), pp. 440–456.

[2] S. CHOPRA, *On the spanning tree polyhedron*, Oper. Res. Lett., 8 (1989), pp. 25–29.

[3] T. H. CORMEN, C. E. LEISERSON, R. L. RIVEST, AND C. STEIN, *Introduction to Algorithms*, MIT Press, Cambridge, MA, 2001.

[4] M. FÜRER AND B. RAGHAVACHARI, *An NC approximation algorithm for the minimum degree spanning tree problem*, in Proceedings of the 28th Annual Allerton Conference on Communication, Control, and Computing, University of Illinois, Urbana-Champaign, IL, 1990, pp. 274–281.

[5] M. FÜRER AND B. RAGHAVACHARI, *Approximating the minimum-degree Steiner tree to within one of optimal*, J. Algorithms, 17 (1994), pp. 409–423.

[6] M. X. GOEMANS AND D. P. WILLIAMSON, *A general approximation technique for constrained forest problems*, SIAM J. Comput., 24 (1995), pp. 296–317.

[7] M. X. GOEMANS AND D. P. WILLIAMSON, *The primal-dual method for approximation algorithms and its application to network design problems*, in Approximation Algorithms for NP-hard Problems, Dorit S. Hochbaum, ed., PWS, Boston, 1997, pp. 144–191.

[8] J. Könemann and R. Ravi, *A matter of degree: Improved approximation algorithms for degree-bounded minimum spanning trees*, in Proceedings of the 32nd ACM Symposium on Theory of Computing, ACM, New York, 2000, pp. 537–546.

[9] J. Könemann and R. Ravi, *A matter of degree: Improved approximation algorithms for degree-bounded minimum spanning trees*, SIAM J. Comput., 31 (2002), pp. 1783–1793.

[10] J. Kruskal, *On the shortest spanning subtree of a graph and the traveling salesman problem*, Proc. Amer. Math. Soc., 7 (1956), pp. 48–50.

[11] R. Ravi, M. V. Marathe, S. S. Ravi, D. J. Rosenkrantz, and H. B. Hunt, *Many birds with one stone: Multi-objective approximation algorithms*, in Proceedings of the 25th ACM Symposium on Theory of Computing, ACM, New York, 1993, pp. 438–447.

[12] R. Ravi, M. V. Marathe, S. S. Ravi, D. J. Rosenkrantz, and H. B. Hunt, *Approximation algorithms for degree-constrained minimum-cost network-design problems*, Algorithmica, 31 (2001), pp. 58–78.

[13] R. Ravi, B. Raghavachari, and P. Klein, *Approximation through local optimality: Designing networks with small degree*, in Foundations of Software Technology and Theoretical Computer Science, Lecture Notes in Comput. Sci. 652, Springer, Berlin, 1992, pp. 279–290.

SIAM J. Comput. © 2005 Society for Industrial and Applied Mathematics
Vol. 34, No. 4, pp. 775–787

# THE COMPLEXITY OF THE UNION OF $(\alpha, \beta)$-COVERED OBJECTS*

ALON EFRAT†

**Abstract.** An $(\alpha, \beta)$-covered object is a simply connected planar region $c$ with the property that for each point $p \in \partial c$ there exists a triangle contained in $c$ and having $p$ as a vertex, such that all its angles are at least $\alpha > 0$ and all its edges are at least $\beta \cdot \mathrm{diam}(c)$-long. This notion extends that of fat convex objects. We show that the combinatorial complexity of the union of $n$ $(\alpha, \beta)$-covered objects of "constant description complexity" is $O(\lambda_{s+2}(n) \log^2 n \log\log n)$, where $s$ is the maximum number of intersections between the boundaries of any pair of given objects, and $\lambda_s(n)$ denotes the maximum length of an $(n, s)$-Davenport–Schinzel sequence. Our result extends and improves previous results concerning convex $\alpha$-fat objects.

**Key words.** fat objects, realistic input models, motion planning

**AMS subject classification.** 65D18

**DOI.** 10.1137/S0097539702407515

## 1. Introduction.

A planar object $c$ is $(\alpha, \beta)$-*covered* if the following conditions are satisfied:

1. $c$ is a compact simply connected set.
2. For each point $p \in \partial c$ we can place a triangle $\Delta$ fully inside $c$, such that $p$ is a vertex of $\Delta$, each angle of $\Delta$ is at least $\alpha$, and the length of each edge of $\Delta$ is at least $\beta \cdot \mathrm{diam}(c)$. We call such a triangle $\Delta$ a *good triangle* for $c$ at $p$.

The notion of an $(\alpha, \beta)$-covered object generalizes that of a convex fat object. A planar convex object $c$ is $\alpha$-*fat* if the ratio between the radii of the disks $s^+$ and $s^-$ is at most $\alpha$, where $s^-$ is the smallest disk containing $c$ and $s^+$ is a largest disk a contained in $c$; see [7]. It is easy to show that if $c$ is an $\alpha$-fat convex object, then it is also an $(\alpha', \beta')$-covered object for appropriate constants $\alpha', \beta'$ that depend only on $\alpha$. Indeed, let $p$ be a point on $\partial c$ and let $q_1, q_2$ be the two antipodal points on $\partial s^+$ such that $|\overline{pq_1}| = |\overline{pq_2}|$. Clearly the isosceles triangle $\Delta q_1 q_2 p$ is at least $\arctan(1/2\alpha)$-fat, and that the length of its edges is at least $\mathrm{diam}(c)/2\alpha$.

In this paper we also make the additional assumption that all the objects under consideration have *constant description complexity*, meaning that each object is a semialgebraic set defined by a constant number of polynomial equalities and inequalities of constant maximum degree. This assumption is critical, since even two convex $m$-gons (for any $m$) can create $2m$ vertices on their union.

The goal of this paper is to obtain sharp bounds for the maximum combinatorial complexity of the union of a collection $\mathcal{C}$ of $n$ $(\alpha, \beta)$-covered objects of constant description complexity for constant parameters $\alpha, \beta > 0$.

There are not too many previous results of this kind. If $\mathcal{C}$ is a collection of $\alpha$-fat triangles,[1] then the complexity of $\bigcup \mathcal{C}$ is $O(n \log\log n)$ (with the constant of proportionality depending on $\alpha$) [15, 16], and this bound improves to $O(n)$ if the triangles are nearly of the same size [2] or are infinite wedges. See also [14] and more

*Received by the editors May 13, 2002; accepted for publication (in revised form) October 11, 2004; published electronically April 29, 2005. A preliminary version of this paper appeared in [6].
http://www.siam.org/journals/sicomp/34-4/40751.html

†Department of Computer Science, University of Arizona, Tucson, AZ 85721 (alon@cs.arizona.edu). This work has been supported by a Rothschild Fellowship and by NSF grant CCR-9623851.

[1]For triangles, there is an equivalent definition of fatness that requires all angles to be at least some fixed constant $\alpha_0$; in [15, 16], this is called $\alpha_0$-fatness.

recently [16] for additional results concerning fat polygons. If $\mathcal{C}$ is a collection of $n$ *pseudodisks* (arbitrary simply connected regions bounded by closed Jordan curves, each pair of whose boundaries intersect at most twice), then the complexity of $\bigcup\mathcal{C}$ is $O(n)$ [13]. Of course, if we drop the fatness condition, the complexity of $\bigcup\mathcal{C}$ can be $\Theta(n^2)$, even for the case of (nonfat) triangles. Even for fat convex objects, some bound on the description complexity of each object must be assumed, or else the complexity of the union might be arbitrarily large.

In three dimensions, there are several relevant results about the union of objects. Agarwal and Sharir [1] showed that if $\mathcal{C}$ is the Minkowski sum of a ball and a set of $n$ disjoint triangles or lines in three dimensions, then the complexity of the boundary of $\bigcup\mathcal{C}$ is $O(n^{2+\varepsilon})$.[2] Among fat objects in three dimensions, one might consider balls and cubes as "basic cases" of fat objects in three dimensions. Pach, Safruti, and Sharir [17] showed that if $\mathcal{C}$ is the union of $n$ wedges, each determined by the intersection of two halfspaces with a dihedral angle at least $\alpha$, then the complexity of their union is $O(n^{2+\varepsilon})$. They also show that if each wedge of $S$ is the intersection of three halfspaces, each pair creating a solid angle at least $\alpha$ and the sum of the three face angles is at least $4\pi/3$, then the complexity of the union of these wedges is $O(n^{2+\varepsilon})$. The constant of proportionality depends on $\varepsilon, \alpha$, and in the latter case, $\gamma$. They also proved that the union of $n$ cubes in three dimensions, whose edge lengths are the same, up to a constant, is also $O(n^{2+\varepsilon})$.

Concerning fat nonpolygonal shapes in the plane, it was shown in [7] that if $\mathcal{C}$ is a collection of $n$ convex $\alpha$-fat objects of constant description complexity, then the complexity of $\bigcup\mathcal{C}$ is $O(n^{1+\varepsilon})$, for any $\varepsilon > 0$, where the constant of proportionality depends on $\varepsilon$, on the fatness parameter, and on the maximum description complexity of the given objects. In an attempt to remove the convexity restriction, the notion of $\kappa$-curviness was introduced in [8]. An object $c$ is *$\kappa$-curved* (for a parameter $\kappa > 0$) if for each point $p$ on its boundary there is a ball $B \subseteq c$ whose radius is at least $\kappa \cdot \mathrm{diam}(c)$, and which contains $p$ on its boundary. It was shown in [8] that if $\mathcal{C}$ is a collection of $n$ *$\kappa$-curved* objects in the plane of constant description complexity, then the complexity of $\bigcup\mathcal{C}$ is $O(\lambda_{s'}(n)\log^2 n)$, where $s'$ is a constant that depends on $\kappa$ and on the description complexity of the objects, and $\lambda_s(n)$ denotes the maximum length of an $(n, s)$-Davenport–Schinzel sequence [18]. This result was recently extended [3], where it was shown that if $\mathcal{C}$ is a collection of $n$ *$\kappa$-curved* objects in $\mathbb{R}^d$, for $d = 3$ and $d = 4$, then the complexity of the boundary of their union is $O(n^{2+\varepsilon})$ in $\mathbb{R}^3$ and $O(n^{3+\varepsilon})$ in $\mathbb{R}^4$. However, the class of $\kappa$-curved objects is rather restricted. For example, an $\alpha$-fat triangle is not a $\kappa$-curved object for any $\kappa$. On the other hand, the notion of $(\alpha, \beta)$-covered objects clearly generalizes the notion of $\kappa$-curved objects, as well as that of fat convex objects.

Let $\mathcal{C}$ be a collection of $n$ $(\alpha, \beta)$-covered objects of constant description complexity in general position; that is, no points lie on the boundaries of three objects, and the boundaries of each pair of objects of $\mathcal{C}$ have at most some constant number, $s$, of intersection points. In addition we assume that $s$ also bounds the number of points at which the boundary of an object in $\mathcal{C}$ is not $C^1$ or $C^2$, as well as the number of inflection points and locally $x$- and $y$-extremal points of any such boundary.

The following is the main result of this paper.

THEOREM 1.1. *Under the above assumptions, the combinatorial complexity of the union of $\mathcal{C}$ is $O(\lambda_{s+2}(n)\log^2 n \log\log n)$.*

[2]In this paper $\varepsilon$ stands for an arbitrary small positive constant.

The proof of Theorem 1.1 is given in the following sections. It is worth mentioning that if all objects of $\mathcal{C}$ are roughly of the same size, then the bound of Theorem 1.1 can be improved to $O(\lambda_{s+2}(n))$; see section 4 for further discussion.

Theorem 1.1, together with the previous works cited above, contribute to the study of the union of planar objects, an area that has many algorithmic applications, such as finding the maximal depth in an arrangement of fat objects (see [10]), hidden surface removal in a collection of fat objects in 3-space [12], point-enclosure queries in a collection of fat objects in the plane [11], and more; see [20] for more applications and other definitions of fat nonconvex objects. Theorem 1.1 both extends these results to the more general class of $(\alpha, \beta)$-covered objects and slightly improves the corresponding complexity bounds.

The contributions of this paper are as follows: (a) the introduction of a new class of "fat" nonconvex objects (namely, $(\alpha, \beta)$-covered objects), which, we believe, captures the characteristics of the input data in most realistic scenes; (b) a sharper bound on the union complexity than the bounds obtained in [7] (bringing them to within a polylogarithmic factor of the best-known lower bounds); and (c) the proof technique, which is much simpler than the analysis given in [7].

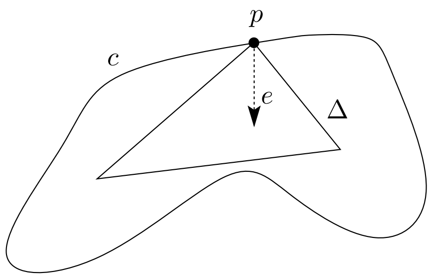


FIG. 1. *The point $p$ is $3\pi/2$-oriented.*

## 2. Preliminaries.

**2. Preliminaries.** Let $\mathcal{C}$ be a collection of $n$ $(\alpha, \beta)$-covered objects, as in the introduction. Let $c \in \mathcal{C}$, and let $p$ be a point on $\partial c$. We say that $p$ is *$\theta$-oriented* if there is a good triangle $\Delta$ for $c$ with $p$ as a vertex, such that the ray $e$ emerging from $p$ at orientation $\theta$ intersects the interior of $\Delta$. In this case we call $\Delta$ a *$\theta$-oriented* triangle at $p$. See Figure 1. Note that $p$ might be $\theta$-oriented for more than one value of $\theta$.

The idea behind our proof is as follows. We define, for each object $c$, a collection of *subobjects* so that every vertex of $\partial \bigcup \mathcal{C}$ is a vertex of the union of the subobjects of all objects $c \in \mathcal{C}$. Moreover, we classify such vertices into $O(\log^2 n)$ disjoint families and argue that the size of each family is only $O(\lambda_{s+2}(n) \log \log n)$.

Let $\Psi$ be the set of orientations $\{\frac{\alpha}{4}, \frac{2\alpha}{4}, \ldots, \frac{\alpha\lceil 8\pi/\alpha \rceil}{4}\}$. We call a triangle $\Delta$ a *$\theta$-critical* triangle at $p$ for $c$ if $\Delta$ is a good triangle at $p$ for $c$, and $\Delta$ is $(\theta - \frac{\alpha}{4})$-oriented at $p$, $\theta$-oriented at $p$, and $(\theta + \frac{\alpha}{4})$-oriented at $p$. We say that a point $p \in \partial c$ is *$\theta$-critical* for $c$ if there exists a $\theta$-critical triangle $\Delta$ at $p$ for $c$. For each $c \in \mathcal{C}$ and each $\theta \in \Psi$ let $\gamma_\theta(c)$ denote the portion of $\partial c$ consisting of $\theta$-critical points. By the constant description complexity assumption made in the introduction, $\gamma_\theta(c)$ consists of at most a constant (depending on $s$ and $\alpha$) connected portions of $\partial c$. We further divide these portions of $\gamma_\theta(c)$ into a constant number of not-too-long subarcs (that might overlap), called *primitive arcs*, or *p-arcs* for short. Each p-arc $\delta$ is required

(i) to be differentiable (that is, there exists a well-defined tangent at each point of the relative interior of $\delta$);

(ii) not to contain in its relative interior any inflection point of $\partial c$;

(iii) to satisfy the property that the difference in the orientations of the tangents at any pair of points of $\delta$ is at most $\pi/t$, where $t > 10$ is a constant to be specified in Claim 2.1 below that depends only on $\alpha$ and $\beta$, and such that $\pi/t < \alpha$;

(iv) to have length no more than $\operatorname{diam}(c)/t'$, where $t'$ is a constant to be specified in Claim 2.1 below and depends only on $\alpha$ and $\beta$.

A p-arc along the boundary of an object $c$ is *convex* if the segment connecting the endpoints of the arc is contained in $c$. Otherwise, we say that the p-arc is *concave.* See Figure 2.

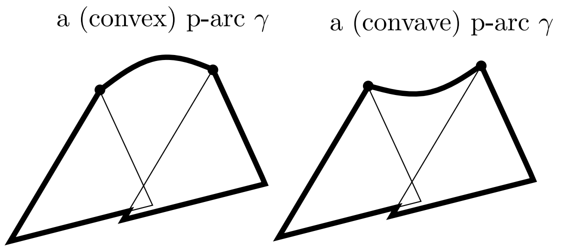


FIG. 2. *Example of convex and concave p-arcs.*

For each $c \in \mathcal{C}$ and for every $\theta \in \Psi$, we place a $\theta$-oriented triangle at each endpoint of every p-arc of $\gamma_\theta(c)$, and we let $P_c$ denote the collection of these triangles.

CLAIM 2.1. *We can choose $t', t$ so that the boundary of each connected component of $c \setminus \bigcup P_c$ contains at most a single p-arc. In other words, there is no path in $c$ that connects two p-arcs of $c$ and does not intersect $\bigcup P_c$.*

*Proof.* Assume $\operatorname{diam}(c) = 1$ (otherwise, we rescale $c$). We say that a point $p$ is *higher than* (resp., *lower than, to the left of, to the right of*) a point $q$, if $p.y \geq q.y$ (resp., $p.y \leq q.y$, $p.x \leq q.x$, $p.x \geq q.x$). Let $\gamma$ be a portion of $\partial c$ satisfying (i), (ii), (iii), and (iv). Assume without loss of generality that the tangent to the rightmost point of $\gamma$ is horizontal. We locate two $3\pi/2$-critical triangles $\Delta'_1, \Delta'_2$, where $p_i$ is a vertex of $\Delta'_i$ (for $i = 1, 2$). Clearly $\Delta'_i$ contains a triangle $\Delta_i \subseteq \Delta'_i$ such that the angle at $p_i$ equals exactly $\alpha/2$, the edge opposite $p_i$ is horizontal (see Figure 3), and the edges adjacent to $p_i$ have length exactly $\beta$ (for $i = 1, 2$). Assume that $p_1$ is to the left of and lower than $p_2$.

We show that if $t' \geq \frac{2}{3\beta \sin(\frac{\alpha}{4})}$ and $t \geq \pi / \arcsin(\beta \cos(\frac{\alpha}{4})/2)$, then $\Delta_1$ and $\Delta_2$ must intersect. Let $u$ and $z$ be the other two vertices of $\Delta_1$ ($u$ to the left of $z$). Let $u'$ be the middle point of $\overline{uz}$ and let $w$ be the middle point of $\overline{u'z}$. Let $w'$ be the point on $\partial \Delta_1$ vertically above $w$, and let $R$ be the axis-parallel rectangle whose two diagonal vertices are $u'$ and $w'$. Let $\mu$ be the left vertex of the lower edge of $\Delta_2$.

Note that if $\mu$ is to the left of $u'$, then clearly $\Delta_1$ and $\Delta_2$ intersect, since they are congruent triangles, $p_2$ is to the right of $p_1$, and $\gamma$ is $x$-monotone. So assume that $\mu$ is to the right of $u'$. To conclude that $\mu \in R$ and thus $\Delta_1$ and $\Delta_2$ intersect, we need to show only that $\mu$ is below and to the left of $w'$. Indeed

$$|\overline{ww'}| = |\overline{p_1 u'}|/2 = |\overline{p_1 z}| \cos\left(\frac{\alpha}{4}\right)/2 = \beta \cos\left(\frac{\alpha}{4}\right)/2 \quad \text{and} \quad |\overline{u'w}| = \beta \sin\left(\frac{\alpha}{4}\right)/2 \; .$$

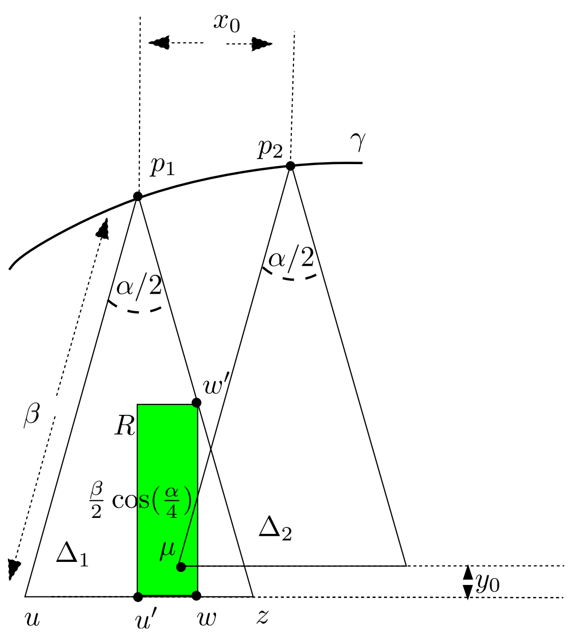


FIG. 3. *Illustration of Claim* 2.1.

Let $x_0 = p_2.x - p_1.x$. Since we set $t' \geq \frac{2}{3\beta\sin(\alpha/4)}$, it follows that

$$x_0 \leq |\gamma| \leq \frac{\text{diam(c)}}{t'} \leq \frac{3}{2}\beta\text{diam(c)}\sin\left(\frac{\alpha}{4}\right) = \frac{3}{2}\beta\sin\left(\frac{\alpha}{4}\right) ,$$

where we make use of the assumption that $\text{diam}(c) = 1$. Hence

$$\mu.x - u'.x \ = \ x_0 - \beta\sin\left(\frac{\alpha}{4}\right) \leq \frac{1}{2}\beta\sin\left(\frac{\alpha}{4}\right) \leq w.x - u'.x,$$

implying that $\mu$ is to the left of $w$. Finally, $\pi/t \leq \arcsin\left(\beta\cos\left(\frac{\alpha}{4}\right)/2\right)$. Hence

$$\mu.y - u'.y \leq |\overline{p_1p_2}|\sin\left(\frac{\pi}{t}\right) \leq \sin\left(\frac{\pi}{t}\right) \leq \beta\cos\left(\frac{\alpha}{4}\right)/2 \leq |\overline{w'w}| ,$$

where we have used the assumption that $|p_1p_2| \leq \text{diam}(c) = 1$. Thus $\mu$ is below $w'$, implying that $\mu \in R$, as claimed. $\square$

We call a maximally connected component of $c \setminus \bigcup P_c$ a *cap*, and the segment connecting the endpoints of its p-arc a *chord*. The union of a cap and the two triangles of $P_c$ adjacent to the endpoints of its p-arc is called a *subobject*. Figure 2 shows two subobjects. If a subobject is not simply connected, we "fill in" its holes and add them to the subobject. The boundary of a subobject consists of a single p-arc and of portions of edges of the good triangles of $P_c$ adjacent to the p-arc's endpoints. Note that the chord of the subobject is not always a part of the subobject. The collection of all subobjects of $c$ that are adjacent to p-arcs that are $\theta$-oriented is denoted by $c^\theta$. See Figure 4. Clearly $c^\theta$ consists of a constant number of subobjects. Let $\mathcal{C}^\theta = \bigcup_{c\in C} c^\theta$.

Fix $\theta \in \Psi$, which we assume for simplicity to be the negative vertical direction; otherwise, rotate the plane. Define a segment tree $\mathcal{T}_\theta$ (see [5] for a discussion of segment trees) over the set of intervals on the $y$-axis obtained by projecting each

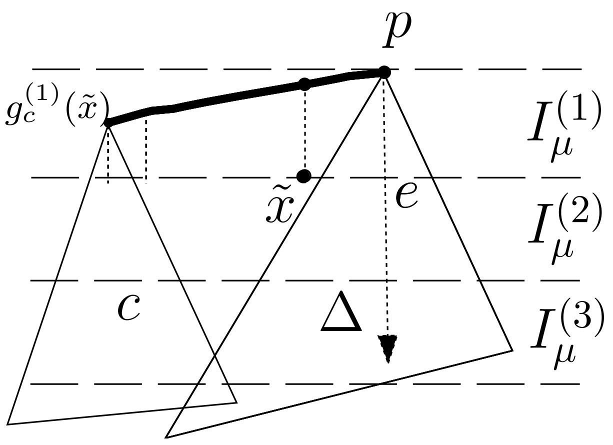


FIG. 4. *A subobject c, a slab and the strips it contains, and the graph of the function* $g_c^{(i)}(x)$.

subobject of $\mathcal{C}^\theta$ on the $y$-axis. Each node $\mu \in \mathcal{T}_\theta$ is associated with a subset $S_\mu \subseteq \mathcal{C}^\theta$ and with a horizontal slab $I_\mu$. See Figure 4. That is, a subobject $c \in C^\theta$ is in $S_\mu$ if and only if its $y$-projection contains the $y$-projection of $I_\mu$ but does not contain the $y$-projection of $I_{\text{parent}(\mu)}$.

Let $\mathcal{L}(\mathcal{T}_\theta, i)$, the $i$th level of $\mathcal{T}_\theta$, denote the collection of nodes of $\mathcal{T}_\theta$ whose distance from the root of $\mathcal{T}_\theta$ is $i$. Fix $\theta_A, \theta_B \in \Psi$ (not necessarily distinct) and levels $i_A$ of $\mathcal{T}_{\theta_A}$ and $i_B$ of $\mathcal{T}_{\theta_B}$. Note that there are $O(\log^2 n)$ quadruples $(\theta_A, \theta_B, i_A, i_B)$ of this form. Define $A = \bigcup_{\mu \in \mathcal{L}(T_{\theta_A}, i_A)} S_\mu$ and $B = \bigcup_{\mu \in \mathcal{L}(T_{\theta_B}, i_B)} S_\mu$. That is, $A$ (resp., $B$) is the collection of subobjects in $S_\mu$ for $\mu$ on the $i_A$th level of $\mathcal{T}_{\theta_A}$ (resp., the $i_B$th level of $\mathcal{T}_{\theta_B}$). Let $U(\theta_A, \theta_B, i_A, i_B)$ denote the set of "mixed" vertices of $\partial \bigcup (A \cup B)$ that lie on $\gamma_{\theta_A}(a)$ for some $a \in A$, and on $\gamma_{\theta_B}(b)$ for some $b \in B$. The next section is dedicated to the proof of the following lemma.

LEMMA 2.1. *The size of* $U(\theta_A, \theta_B, i_A, i_B)$ *is* $O(\lambda_{s+2}(n) \log\log n)$.

The proof of Theorem 1.1 follows immediately from Lemma 2.1. Indeed, for each vertex $v \in \partial \bigcup \mathcal{C}$ which lies on the boundaries of objects $a, b \in \mathcal{C}$ there is a direction $\theta_A \in \Psi$ (resp., $\theta_B \in \Psi$) such that $v$ is $\theta_A$-critical for $a$ at $v$ (resp., $\theta_B$-critical for $b$ at $v$). Hence there is a subobject $a' \subset a$ containing $v$ on its boundary, and a node $\mu_a \in \mathcal{T}_{\theta_A}$ with $a' \in S_{\mu_{a'}}$. Hence $v \in \partial \bigcup A$, where $A = \bigcup_{\mu \in \mathcal{L}(T_{\theta_A}, i_A)} S_\mu$, and $\mathcal{L}(T_{\theta_A}, i_A)$ is the level of $\mathcal{T}_{\theta_A}$ containing $a'$. Similarly there exist $\theta_B$ and $i_B$ such that $v \in \partial \bigcup B$, where $B = \bigcup S_{\mu \in \mathcal{L}(T_{\theta_B}, i_B)} S_\mu$. Thus $v \in U(\theta_A, \theta_B, i_A, i_B)$. Since there are only $O(\log^2 n)$ quadruples $(\theta_A, \theta_B, i_A, i_B)$, Theorem 1.1 follows immediately from Lemma 2.1.

**3. Proof of Lemma 2.1.** We fix a quadruple $(\theta_A, \theta_B, i_A, i_B)$, as above. We assume that $\theta_A = 3\pi/2$ (the negative $y$-direction) by rotating the plane if necessary. Let $\mu$ be a node in the $i_A$th level of $\mathcal{T}_{3\pi/2}$, and let $I_\mu$ be the horizontal slab associated with $\mu$. Let $c$ be a subobject of $S_\mu$, let $p \in \gamma_{3\pi/2}(c) \cap I_\mu$, and let $\Delta$ be a $(3\pi/2)$-oriented triangle of $c$ at $p$. Note that $\Delta$ is an $\alpha$-fat triangle, and the length of the $y$-span of $c$ is at least the width of $I_\mu$. Hence there is a constant integer $h$ (that depends only on $\alpha$ and $\beta$), such that the length of the $y$-span of each edge of $\Delta$ is at least $1/h$ times the width of $I_\mu$. We divide $I_\mu$ into $h$ interior-disjoint *strips*, $I_\mu^{(1)}, \ldots, I_\mu^{(h)}$, of equal width. Thus, $p$ lies in a different strip from the other two vertices of $\Delta$; see Figure 4. Therefore, for each strip $I_\mu^{(i)}$, we can express all points of $I_\mu^{(i)} \cap \gamma_{3\pi/2}(c)$ as a graph

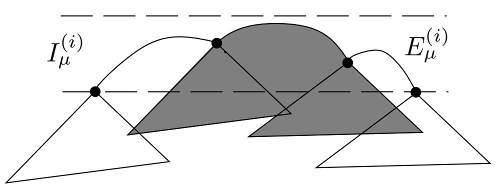


FIG. 5. *The upper envelope $E_\mu^{(i)}$ and some new subobjects defined by its vertices. One of the subobjects is shaded.*

$g_c^{(i)}(\tilde{x})$ of a (partial) function defined on the lower boundary of the strip $I_\mu^{(i)}$, with the property that the segment connecting the points $\tilde{x}$ to $g_c^{(i)}(\tilde{x})$ is fully contained inside $c$.

For each strip $I_\mu^{(i)}$, consider the upper envelope $E_\mu^{(i)}$ (see Figure 5) of the functions $g_c^{(i)}(\tilde{x})$ for $c \in \mathcal{S}_\mu$. Let $E_\mu$ denote the union of these upper envelopes for all strips of $I_\mu$, and let $E_A$ denote the union of all these envelopes taken over all nodes $\mu$ in the $i_A$th level of $\mathcal{T}_{\theta_A}$. Repeat the same analysis for $\theta_B$, and obtain a corresponding union $E_B$ of upper envelopes (relative to the $\theta_B$-direction).

Let $v$ be a vertex of $E_A$ incident to the boundaries of subobjects $c_1, c_2 \in A$. We add to $P_{c_1}$ a $\theta_A$-critical triangle for $c_1$ at $v$, and to $P_{c_2}$ a $\theta_A$-critical triangle for $c_2$ at $v$ (it is possible that these are similar triangles with a common vertex and overlapping edges). For each $c \in \mathcal{S}_\mu$ and each p-arc $\gamma$ of $c$ we also add $\theta_A$-critical triangles at each point where $\gamma$ crosses a boundary of a strip of $I_\mu$. We further refine the splitting of arcs into p-arcs, so that no p-arc $\gamma$ contains a vertex of a new triangle in the p-arc's interior. Subobjects are split as well, so that each subobject contains exactly one new p-arc on its boundary. Observe that now each p-arc is contained in at most one strip of $I_\mu$.

We next remove from $A$ all subobjects that do not participate in $E_A$. Thus each p-arc of a remaining subobject of $A$ is fully contained in $E_A$, and also fully contained in a single strip of some $I_\mu$. Analogously, we restructure the subobjects and p-arcs for $B$, the collections $\{P_b\}_{b \in B}$ and the union of envelopes $E_B$. We list several important properties of this construction:

(A1) Two p-arcs of $A$ either are disjoint or intersect only at their endpoints. Moreover, a p-arc $\gamma$ of a subobject $a_1 \in A$ may intersect the boundary of a different subobject $a_2 \in A$ only at an endpoint of $\gamma$ or at a point of $\partial \bigcup P_{a_2}$. Similar properties hold for $B$.

(A2) A necessary condition for a vertex $v$ to belong to $U(\theta_A, \theta_B, i_A, i_B)$ is that $v$ lies on $E_A$ and on $E_B$.

(A3) The complexities of $E_A$ and of $E_B$ are each $O(\lambda_{s+2}(n))$; see [18] for a discussion on the complexity of an upper envelope.

Recall that the parameter $t$ used in property (iii) satisfies $\pi/t \le \alpha$. We next state a slightly modified version of a lemma that appeared in [7].

LEMMA 3.1 (Efrat and Sharir [7]). *Let $K_a$ be the portion of a cap of some subobject $a \in \mathcal{C}^\theta$, enclosed between its convex p-arc $\gamma_a$ and its chord $e_a$. Let $\Delta_b \in P_b$ be a good triangle for some object $b \in \mathcal{C}$, such that an edge $e_b$ of $\Delta_b$ crosses $\gamma_a$. Then one of the following cases must occur:*

(i) *$e_a$ crosses $\partial \Delta_b$ (as in Figure* 6(i)*).*

(ii) *$K_a$ contains a vertex of $\Delta_b$ that is an endpoint of $e_b$ (as in Figure* 6(ii)*).*

(iii) *$\Delta_b$ contains a vertex of $K_a$ (as in Figure* 6(iii)*).*

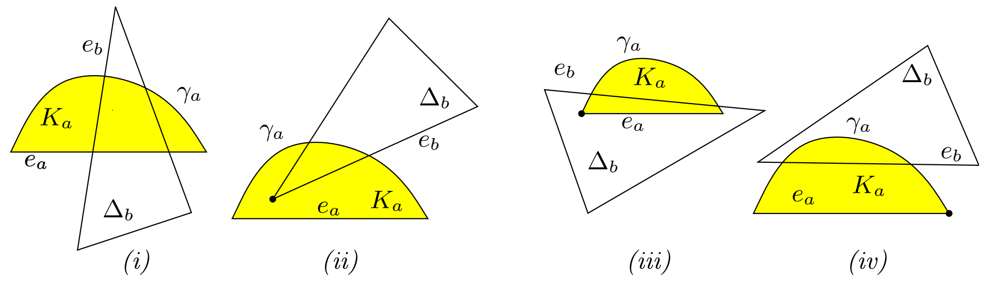


FIG. 6. *Illustrating the various cases in Lemma* 3.1.

(iv) *$\partial K_a$ and $\partial \Delta_b$ cross exactly twice, at two points that lie on $\partial a$ and on $e_b$, and $e_a$ is disjoint from $\Delta_b$ (as in Figure* 6(iv)*).*

So, for example, it cannot happen that $\gamma_a$ intersects two edges of $\Delta_b$, each edge in two points, and all vertices of $\Delta_b$ lie in the same side of the line containing $e_a$.

*Proof.* The proof is essentially identical to the one given in [7], with obvious modifications that are left to the reader. □

Consider the collections $P_A = \bigcup_{a \in A} P_a$, $P_B = \bigcup_{b \in B} P_b$. The result of [15, 16] implies that the complexity of $\bigcup P_A$, of $\bigcup P_B$, and of $\bigcup(P_A \cup P_B)$ are each $O(\lambda_{s+2}(n) \log\log n)$, as each triangle in these collections is an $\alpha$-fat triangle. The constants of proportionality depend on $\alpha$.

Our strategy in proving Lemma 2.1 is to proceed in several steps. First, we define below sets of new types of vertices, namely, $UP(A)$ and $UP(B)$. We obtain bounds on the cardinality of these sets, by charging (roughly speaking) each vertex $v$ of these sets to a vertex $u$ of $\bigcup(P_A \cup P_B)$, in a way that $u$ is not charged more than a constant number of times. Later on, we charge each vertex $w$ of $U(\theta_A, \theta_B, i_A, i_B)$ to a vertex of $\bigcup(P_A \cup P_B)$, or a vertex of $UP(A)$ or $UP(B)$.

We define $UP(A)$ as the set of all vertices of $E_A \cap \partial \bigcup(A \cup P_B)$. We define $UP(B)$ symmetrically, interchanging $A$ and $B$.

LEMMA 3.2. *The number of vertices of $UP(A)$ and of $UP(B)$ is $O(\lambda_{s+2}(n) \log\log n)$.*

*Proof.* It suffices to prove the lemma for $UP(B)$. Consider a p-arc $\gamma_b = \gamma_{\theta_B}(b)$ containing a vertex $v$ of $UP(B)$, defined by the intersection of $\gamma_b$ and an edge $e$ of $\partial \bigcup(P_A \cup P_B)$. Let $\Delta$ be the triangle adjacent to $e$ (note that $e$ might be a portion of an edge of $\Delta$) and let $\hat{e}$ be the edge of $\Delta$ containing $e$.

Let $u_1$, $u_2$ be the endpoints of $\gamma_b$; see Figure 7. Assume again that $\theta_B = 3\pi/2$, so the slabs of $\mathcal{T}_{\theta_B}$ are horizontal, and suppose $u_1$ is to the left of $u_2$. Let $\mu$ be the node of $\mathcal{T}_{\theta_B}$, on the $i_B$th level, associated with the subobject $b$ containing $v$ on its boundary. Let $t_1$ and $t_2$ be the triangles of $P_b$, which are $\theta_B$-critical for $b$ at $u_1$ and at $u_2$. Let $F = F(\gamma_b)$ be the axis-parallel rectangle formed by intersecting $I_\mu^{(i)}$ with the vertical strip spanned by $\gamma_b$ (the shaded area in Figure 7). Clearly $v$ lies in $F$. Note that if $\gamma_b, \gamma_{b'}$ are two distinct p-arcs of $A$, then the rectangles $F(\gamma_b)$ and $F(\gamma_{b'})$ are interior disjoint.

If $\Delta \cap \gamma_b$ fully contains one of the two portions of $\gamma_b$ connecting $v$ to one of the endpoints of $\gamma_b$, we charge $v$ to this endpoint. Since the number of endpoints is $O(\lambda_{s+2}(n))$ and each can be charged at most twice, from each direction, the number of vertices $v$ of this type is within the asserted bound. So assume this is not the case.

Recall that $\gamma_b$ is either concave or convex; so $e$ intersects $\gamma_b$ either once or twice. We call $e$ a *long* edge if both its endpoints are outside $F$; otherwise, $e$ is a *short* edge.

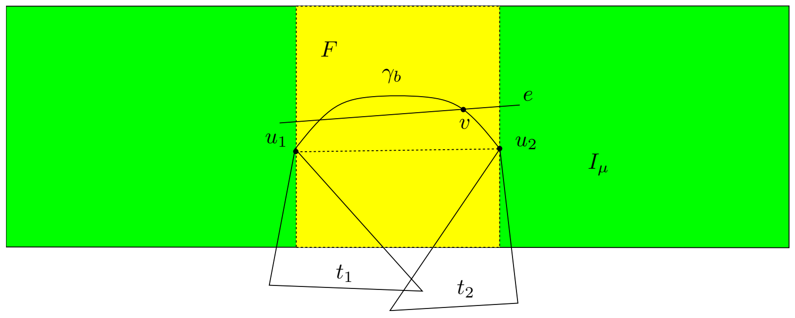


FIG. 7. *Illustrating the proof of Lemma* 3.2*; e is long.*

If $e$ is short, we charge $v$ to an endpoint of $e$ inside $F$ (which in turn can be charged at most twice); again, the number of such endpoints is within the asserted bound. So let us assume that $e$ is long.

If $e$ intersects $\gamma_b$ once, then $e$ must intersect $t_1$, $t_2$, or some other triangle of $P_A \cup P_B$, at a point inside $F$ and on $\partial \bigcup(P_A \cup P_B)$. Hence by following $e$ inward into $b$, we must reach a vertex $x$ of $\bigcup(P_A \cup P_B)$ that we can charge to $v$. So assume that $e$ intersects $\gamma_b$ twice. If $\gamma_b$ is concave, then if we trace $e$ from $v$ into $b$, we reach a vertex of $\partial \bigcup(P_A \cup P_B)$, to which we can charge $v$. So we may assume that $\gamma_b$ is convex, as depicted in Figure 7.

Let $z$ be the vertex of $\Delta$ that lies opposite $e$. We say that $e$ is *special* if $z$ lies inside $F$. We charge $v$ in this case to $z$, and $z$ could be charged at most twice (by the two vertices which lie on the edge of $\Delta$, and contained in $F$). Hence it suffices to consider the case where $e$ is nonspecial and long.

Applying Lemma 3.1 to the edge $\hat{e}$ of $\Delta$, and the appropriate cap portion, we see that if any of the cases (i)–(iii) arises, we can charge $v$ to a vertex of $\bigcup(P_A \cup P_B)$ inside the cap, as done above. So we may assume that case (iv) arises.

We now claim that the number of long nonspecial edges $e_1, \ldots, e_l$ of $\bigcup(P_A \cup P_B)$ incident to vertices of $U(\theta_A, \theta_B, i_A, i_B)$ on $\gamma_b$, satisfying property (iv) of Lemma 3.2, is at most two. Indeed, let $\Gamma(e_i)$ be the portion of $\gamma_b$ spanned between its two intersection points with $e_i$.

It is impossible that $\Gamma(e_i)$ and $\Gamma(e_j)$ intersect. Indeed if $\Gamma(e_i) \cap \Gamma(e_j) \neq \emptyset$, but neither $\Gamma(e_i) \subseteq \Gamma(e_j)$ nor $\Gamma(e_j) \subseteq \Gamma(e_j)$, then $e_i$ and $e_j$ must intersect inside $F$ (by the convexity of $\gamma$), and thus they are not long. On the other hand, it is impossible that one of them, say $\Gamma(e_i)$, is fully contained in $\Gamma(e_j)$, since they both satisfy property (iv), implying that then $\Gamma(e_i)$ is fully contained inside the triangle adjacent to $e_j$, and preventing $e_i$ from determining vertices of $U(\theta_A, \theta_B, i_A, i_B)$ on $\gamma_b$. Moreover, if $\Gamma(e_i)$ and $\Gamma(e_j)$ are disjoint (see Figure 8), then it is easily verified that $e_i$ and $e_j$ intersect different pairs of edges of $F$; thus there are only two such pairs that intersect $\gamma$. This concludes the proof of the Lemma 3.2. $\square$

We can now return to the proof of Lemma 2.1.

*Proof of Lemma* 2.1. Let $v$ be a vertex of $U = U(\theta_A, \theta_B, i_A, i_B)$, incident to a p-arc $\gamma_a$ and to another p-arc $\gamma_b$, for some $a \in A$, $b \in B$. Let $I_\mu$ (for $\mu \in \mathcal{T}_{\theta_B}$) be the strip containing $b$ and assume again that $\theta_B = 3\pi/2$; otherwise, rotate the plane. As in Lemma 3.2, let $F$ be the rectangle formed by the intersection of $I_\mu$ with the vertical strip spanned by $\gamma_b$. We call $\gamma_a$ *favorable* if $\gamma_a \cap F$ contains either an endpoint of $\gamma_a$, or a locally highest point, lowest point, rightmost point, or leftmost point of $\gamma_a$. Note

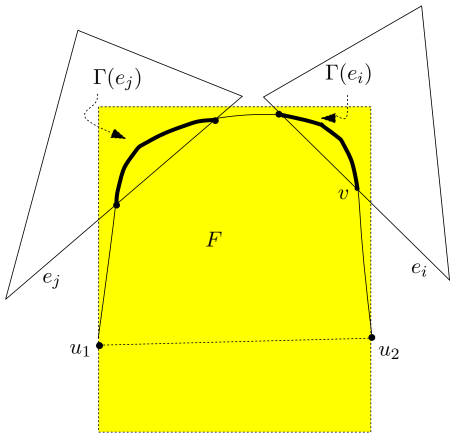


FIG. 8. *If* $\Gamma(e_i)$ *and* $\Gamma(e_j)$ *are disjoint, then* $e_i$ *and* $e_j$ *cross different pairs of edges of* $F$*.*

that if $\gamma_a$ is favorable, we can charge $v$, and the rest of the (at most $s-1$) vertices of $U \cap \gamma_a \cap \gamma_b$ to one of the extreme points listed above, since there is only a constant number of them on each object of $\mathcal{C}$.

Let $\mu_a$ and $\mu_b$ be the normals at $v$ to $\gamma_a$ and $\gamma_b$, pointing into $a$ and $b$, respectively. Let $\phi$ be the smaller angle between $\mu_a$ and $\mu_b$. Let $\phi_0 < \pi/10$ denote the maximal turning angle of any p-arc. See property (iii).

We distinguish among three cases:

*Case* (i). $\phi_0 \le \phi < \pi - 2\phi_0$ (see Figure 9(i)). Clearly, in this case $\gamma_a$ and $\gamma_b$ have at most one intersection point, which must be $v$ itself. Indeed, construct a line $\ell$ that passes through $v$ and forms angles $\phi$ and $-\phi/2$ with $\gamma_a$ and $\gamma_b$, respectively. Since neither $\gamma_a$ nor $\gamma_b$ can turn by more than $\phi$, it follows that, except at $v$, $\ell$ is disjoint from both $\gamma_a$ and $\gamma_b$. Hence, except at $v$, $\ell$ separates $\gamma_a$ from $\gamma_b$.

We follow $\gamma_a$ from $v$ in the direction in which it enters $b$. Since $\gamma_a$ has entered the cap of $b$ bounded by $\gamma_b$ and it does not intersect $\gamma_b$ again, it either ends within the cap or meets a triangle in $P_A \cup P_B$. In either case we can charge $v$ to this endpoint or intersection point. (Note that in the latter case, this intersection must be a vertex of $UP(A)$, whose number was bounded in Lemma 3.2.)

*Case* (ii). $\phi \ge \pi - 2\phi_0$ (see Figure 9(ii)). Without loss of generality, assume that the situation is as shown in Figure 9(ii). That is, $a$ lies above $\gamma_a$ near $v$ and $b$ lies below $\gamma_b$ near $v$, and as we trace $\gamma_a$ and $\gamma_b$ to the left, each of them enters into the other object. If we reach in any of these tracings a point on $\bigcup(P_A \cup P_B)$, then this is a vertex of either $UP(A)$ or $UP(B)$, to which we can charge $v$. So assume this is not the case. Hence, $\gamma_a$ and $\gamma_b$ must intersect again.

If one of the connected components of $\gamma_a \cap F$ intersects the same edge of $F$ twice, then it must contain an extreme point (lowermost, uppermost, rightmost, or leftmost) which is inside $F$, and hence $\gamma_a$ is favorable. We charge the $s$ or fewer intersection points of $\gamma_a$ and $\gamma_b$ inside $F$ to this extreme point. So again assume this is not the case. Assume that $\gamma_1, \dots, \gamma_l$ are all the nonfavorable p-arcs belonging to (not necessarily distinct) respective subobjects $a_1, \dots, a_l$ of $A$, that each defines a vertex $v_i$ of $U(\theta_A, \theta_B, i_A, i_B)$ that lies on $\gamma_b$, and that each $v_i$ is of the type discussed in case (ii) of Lemma 2.1. For each $i$, define $\Gamma(\gamma_i)$ as the portion of $\gamma_b \cap a_i$ incident to $v_i$.

Note that $\gamma_i$ cannot intersect $\gamma_j$ inside $F$ (for any $1 \le i < j \le l$) because each $\gamma_i$ is

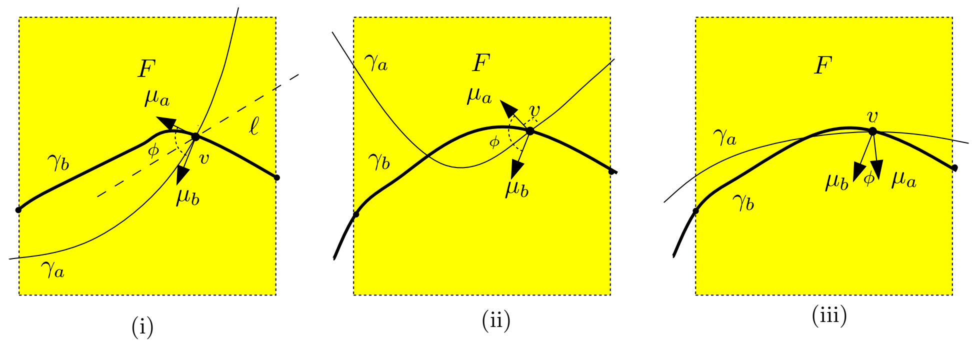


FIG. 9. *Illustration of the proof of Lemma* 2.1.

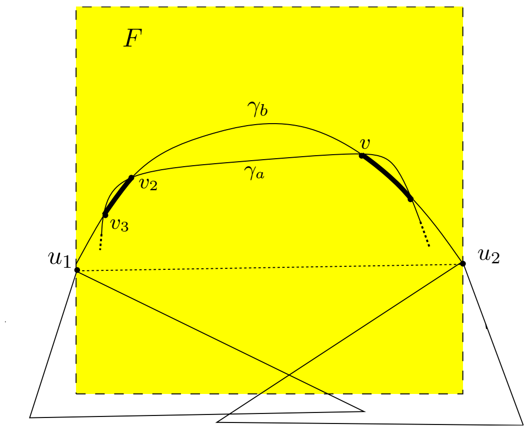


FIG. 10. *The third case of the proof of Lemma* 2.1.

a nonfavorable p-arc, and the p-arcs of $A$ (resp., $B$) intersect only at their endpoints. It is not hard to show that there can be at most 2 nonfavorable p-arcs $\gamma_i$, $\gamma_j$ of this type, such that $\Gamma(\gamma_i) \cap \Gamma(\gamma_j) = \emptyset$. Similarly there is no pair $\gamma_i, \gamma_i$ such that $\Gamma(\gamma_i) \cap \Gamma(\gamma_j)$ partially overlap (that is, $\Gamma(\gamma_i) \cap \Gamma(\gamma_j) \neq \emptyset$ but neither $\Gamma(\gamma_i) \subseteq \Gamma(\gamma_j)$ nor $\Gamma(\gamma_j) \subseteq \Gamma(\gamma_i)$). On the other hand, it is impossible that $\Gamma(\gamma_i) \subseteq \Gamma(\gamma_j)$, since in this case $\Gamma(\gamma_i) \subseteq a_j$ cannot be adjacent to a vertex of $U(\theta_A, \theta_B, i_A, i_B)$.

*Case* (iii). $\phi < \phi_0$ (see Figure 9(iii)). Observe that both $\gamma_a$ and $\gamma_b$ are $y$-monotone inside $F$. We follow $\gamma_a$ from $v$ in the direction inward $b$—say to the left (see Figure 10). If we reach a triangle of $P_A \cup P_B$, then this is a vertex $w$ of $UP(A)$ that we charge. Thus we assume that we reach another vertex $v_2$ on $\gamma_b \cap \gamma_a$, to the left of $v$. By property A1 we are guaranteed that we have not entered a subobject $a' \in A$ so far in the portion of $\gamma_a$ between $v$ and $v_2$. Hence we deduce that $v_2$ is also a vertex of $U(\theta_A, \theta_B, i_A, i_B)$. We continue following the portion of $\gamma_b$ to the left of $v_2$. As before if we do not reach a vertex of $UP(A)$, we reach a vertex $v_3$ which is also a vertex of $U(\theta_A, \theta_B, i_A, i_B)$. We can repeat this process at most $s$ times, discovering the vertices $v, v_2, v_3, \ldots$. This process must terminate when we reach a vertex $w$ of $UP(A)$ or of $UP(B)$ (which may be the left endpoint of $\gamma_b$). Note that by the manner in which we reach $w$, it is obvious that it is charged only to the at most $s$ vertices $v, v_2, v_3, \ldots$, which concludes the proof of Lemma 2.1. ∎

## 4. Concluding remarks.

1. The bound of Theorem 1.1 improves if the objects of $\mathcal{C}$ have roughly the same size. Assume that there are constants $d, \kappa$, such that $d \leq \mathrm{diam}(c) \leq \kappa d$ for each object of $c \in \mathcal{C}$. Then the bounds of Theorem 1.1 improve to $O(\lambda_{s+2}(n))$, with constant of proportionality depending on $\kappa$. This follows by modifying the preceding proof, and we will comment on only a few of the less trivial modifications that are required. In particular we do not need to use segment trees to determine the strips $I_\mu^{(j)}$, since these strips have the same width.
   For each orientation $\theta_A \in \Psi$, we divide the plane into infinite parallel strips of width $\mu d$ (for a sufficiently small constant $\mu$ that depends on $s, \alpha, \kappa$, and $\beta$), orthogonal to the $\theta_A$ direction, such that (as above) if $\Delta$ is a $\theta_A$-critical triangle for an object $c$ at a point $p \in \partial c$, then the other two vertices of $\Delta$ do not lie in the strip containing $p$. We define $A$ as the collection of subobjects incident to p-arcs of $\gamma_{\theta_A}(c)$ over all $c \in \mathcal{C}$. The definitions of $B$ and of all the other notation used in the proof are analogous. We also use the fact that all the oriented triangles in $P_A$ and $P_B$ are roughly of the same size, and thus the complexity of their union is only $O(n)$, as shown in [2].
2. The definition of an $(\alpha, \beta)$-covered object is not the first attempt to define fatness for nonconvex objects. In [20], van der Stappen gives the following definition to fatness. An object $c \subseteq \mathbb{R}^d$ with bounded-description complexity is $\delta$-fat (for $0 < \delta < 1$) if for each $d$-dimensional ball $B$, the center of which is inside $c$ but which does not contain $c$ completely, the volume of $B \cap c$ is at least $\delta$ times the volume of $B$. Two questions naturally arise: (i) What is the relation between the class of $(\alpha, \beta)$-covered objects and $\delta$-fat objects? (ii) Can one show a bound on the complexity of the union of $\delta$-fat objects? Recently van der Stappen [19] answered both questions: He showed that the definition of a $\delta$-fat object is more general than the definition of an $(\alpha, \beta)$-objects by showing that each $(\alpha, \beta)$-covered object is also a $\delta$-fat object for an appropriate parameter $\delta$ (that depends on $\alpha$ and $\beta$). He also answered the second question by presenting a construction showing that the boundary of the union of $n$ $\delta$-fat objects can have $\Omega(n^2)$ vertices, implying that $\delta$-fatness does not suffice to provide subquadratic union complexity in the plane.

**Acknowledgments.** We wish to thank Micha Sharir for helpful discussions and many useful ideas regarding the results of this paper. Thanks also to anonymous referees for numerous important remarks.

## REFERENCES

[1] P. Agarwal and M. Sharir, *Pipes, cigars, and Kreplach: The union of Minkowski sums in three dimensions*, Discrete Comput. Geom., 24 (2000), pp. 645–685.

[2] P. Agarwal, M. Katz, and M. Sharir, *Computing depth orders for fat objects and related problems*, Comput. Geom., 5 (1995), pp. 187–206.

[3] B. Aronov, A. Efrat, V. Koltun, and M. Sharir, *On the union of $\kappa$-round objects in three and four Dimensions*, in Proceedings of the 20th Annual Symposium on Computational Geometry, 2004, pp. 383–390.

[4] B. Chazelle, H. Edelsbrunner, L. Guibas, and M. Sharir, *Algorithms for bichromatic line segment problems and polyhedral terrains*, Algorithmica, 11 (1994), pp. 116–132.

[5] M. de Berg, M. van Kreveld, M. H. Overmars, and O. Schwarzkopf, *Computational Geometry: Algorithms and Applications*, 2nd ed., Springer-Verlag, Berlin, 2000.

[6] A. Efrat, *The complexity of the union of $(\alpha, \beta)$-covered objects*, in Proceedings of the 15th Annual Symposium on Computational Geometry, 1999, pp. 134–142.

[7] A. Efrat and M. Sharir, *The complexity of the union of fat objects in the plane*, Discrete Comput. Geom., 23 (2000), pp. 171–189.

[8] A. Efrat and M. Katz, *On the union of $\alpha$-curved objects*, Comput. Geom., 14 (1999), pp. 241–254.

[9] A. Efrat, M. Katz, F. Nielsen, and M. Sharir, *Dynamic data structures for fat objects and their applications*, Comput. Geom., 15 (2000), pp. 215–227.

[10] A. Efrat, M. Sharir, and A. Ziv, *Computing the smallest $k$-enclosing circle and related problems*, Comput. Geom., 4 (1994), pp. 119–136.

[11] M. Katz, *3-D vertical ray shooting and 2-D point enclosure, range searching, and arc shooting amidst convex fat objects*, Comput. Geom., 8 (1997), pp. 299–316.

[12] M. Katz, M. Overmars, and M. Sharir, *Efficient output sensitive hidden surface removal for objects with small union size*, Comput. Geom., 2 (1992), pp. 223–234.

[13] K. Kedem, R. Livne, J. Pach, and M. Sharir, *On the union of Jordan regions and collision-free translational motion amidst polygonal obstacles*, Discrete Comput. Geom., 1 (1986), pp. 59–71.

[14] M. van Kreveld, *On fat partitioning, fat covering, and the union size of polygons*, in Proceedings of the 3rd Workshop Algorithms Data Structure, Lecture Notes in Comput. Sci. 709, Springer-Verlag, Berlin, 1993, pp. 452–463.

[15] J. Matoušek, J. Pach, M. Sharir, S. Sifrony, and E. Welzl, *Fat triangles determine linearly many holes*, SIAM J. Comput., 23 (1994), pp. 154–169.

[16] J. Pach and G. Tardos, *On the boundary complexity of the union of fat triangles*, SIAM J. Comput., 31 (2002), pp. 1745–1760.

[17] J. Pach, I. Safruti, and M. Sharir, *The complexity of the union of cubes in three dimensions*, Discrete Comput. Geom., to appear.

[18] M. Sharir and P. K. Agarwal, *Davenport-Schinzel Sequences and Their Geometric Applications*, Cambridge University Press, New York, 1995.

[19] A. F. van der Stappen, *private communication*, 1999.

[20] A. F. van der Stappen, *Motion Planning Amidst Fat Obstacles*, Ph.D. Dissertation, Utrecht University, 1994.

SIAM J. COMPUT.
Vol. 34, No. 4, pp. 788–802

# STOCHASTIC MACHINE SCHEDULING WITH PRECEDENCE CONSTRAINTS[*]

MARTIN SKUTELLA[†] AND MARC UETZ[‡]

**Abstract.** We consider parallel, identical machine scheduling problems, where the jobs are subject to precedence constraints and release dates, and where the processing times of jobs are governed by independent probability distributions. Our objective is to minimize the expected value of the total weighted completion time. Building upon a linear programming relaxation by Möhring, Schulz, and Uetz [*J. ACM*, 46 (1999), pp. 924–942] and a delayed list scheduling algorithm by Chekuri et al. [*SIAM J. Comput.*, 31 (2001), pp. 146–166], we derive the first constant-factor approximation algorithms for this model.

**Key words.** approximation algorithms, stochastic scheduling, parallel machines, precedence constraints, release dates, list scheduling algorithms, LP-relaxation

**AMS subject classifications.** 68M20, 68Q25, 68W25, 68W40, 90B36, 90C05

**DOI.** 10.1137/S0097539702415007

## 1. Introduction.

This paper addresses stochastic parallel machine scheduling problems with the objective of minimizing the expected value of the total weighted completion time. Machine scheduling problems have attracted researchers for decades since such problems play an important role in various applications from the areas of operations research, management science, and computer science. The total weighted completion time objective is of particular importance in parallel processing, where many jobs are to be scheduled on a limited number of machines and a good average performance is desired. Prominent examples for such a scheduling situation are problems that arise, e.g., in compiler optimization [4] and in parallel computing [2]. The main characteristic of *stochastic* scheduling problems is the fact that the processing times of the jobs may be subject to random fluctuations. Hence, the effective processing times are not known with certainty in advance. This characteristic is of particular practical relevance in many applications.

**Problem definition.** Denote by $V = \{1, \dots, n\}$ a set of jobs which must be scheduled on $m$ parallel, identical machines. Each machine can handle only one job at a time, and the jobs can be scheduled on any of the machines. Once the processing of a job is started on one machine, it must be processed without preemption on this machine. Precedence constraints are given by an acyclic digraph $G = (V, A)$, where

[*]Received by the editors September 20, 2002; accepted for publication (in revised form) August 28, 2004; published electronically April 29, 2005. This work was done while both authors were with Technische Universität Berlin, Germany. An extended abstract [20] appeared in the *Proceedings of the* 12*th Annual ACM-SIAM Symposium on Discrete Algorithms*, SIAM, Philadelphia, ACM, New York, 2001, pp. 589–590.

http://www.siam.org/journals/sicomp/34-4/41500.html

[†]Fachbereich Mathematik, Universität Dortmund, Vogelpothsweg 87, 44227 Dortmund, Germany (martin.skutella@uni-dortmund.de). The research of this author was supported in part by the EU Thematic Networks APPOL I+II, Approximation and Online Algorithms, IST-1999-14084 and IST-2001-30012, and by the DFG Research Center "Mathematics for key technologies: Modelling, simulation and optimization of real-world processes."

[‡]Faculty of Economics and Business Administration, Quantitative Economics, Universiteit Maastricht, NL-6200 MD Maastricht, The Netherlands (m.uetz@ke.unimaas.nl). The research of this author was supported in part by German-Israeli Foundation for Scientific Research and Development (GIF) grant I 246-304.02/97 and by Deutsche Forschungsgemeinschaft (DFG) grant Mo 446/3-4.

any arc $(i, j) \in A$ restricts the start time of job $j$ to be not earlier than the completion time of job $i$. We consider problems with and without release dates $r_j$ for the jobs, with the intended meaning that job $j$ must not start earlier than $r_j$. In the classical (deterministic) setting, the objective is to minimize the total weighted completion time $\sum_{j\in V} w_j\, C_j$, where $w_j$ is a nonnegative weight and $C_j$ denotes the completion time of job $j$. In the stochastic model, it is assumed that the processing time $p_j$ of a job $j$ is not known in advance. It becomes known only upon completion of the job. However, the distribution of the corresponding random variable $P_j$ is given beforehand. Let $P = (P_1, \dots, P_n)$ denote the vector of random variables for the processing times, and denote by $p = (p_1, \dots, p_n)$ a particular realization of the processing times. By $\mathrm{E}[P_j]$ we denote the expected processing time of a job $j$. We assume throughout that the processing times of the jobs are stochastically independent. In the classical $\alpha\,|\,\beta\,|\,\gamma$ notation of Graham et al. [9], the problem of minimizing the expected total weighted completion time can be denoted by $\mathrm{P}|\,\mathit{prec}, r_j|\mathrm{E}\,[\sum w_j\, C_j]$. Here, P stands for the parallel machine environment, *prec* and $r_j$ the existence of precedence constraints and release dates, respectively, and $\mathrm{E}\,[\sum w_j\, C_j]$ the objective of minimizing the expected total weighted completion time.

**Dynamic view on stochastic scheduling.** The twist from deterministic to stochastic processing times changes the nature of the scheduling problem considerably. The solution of a stochastic scheduling problem is no longer a simple schedule, but a so-called *scheduling policy.* We adopt the notion of scheduling policies as defined by Möhring, Radermacher, and Weiss [14]. In the following, we briefly summarize what that means.

Apart from the data that specifies the input of the problem, the *state* of the system at any time $t \geqslant 0$ is determined by the time $t$ itself, as well as the (conditional) probability distributions of the jobs' processing times. At any time $t > 0$, the state thus depends on the observed *past* up to time $t$. This includes the start and completion times of the jobs already completed by $t$, together with the start times of the jobs in process at time $t$. The *action* of a scheduling policy at time $t$ is given by a set of jobs $B(t) \subseteq V$ that is started at $t$, together with a tentative next decision time $t_{\mathrm{tent}} > t$. The tentative decision time $t_{\mathrm{tent}}$ is the latest point in time when the next action of the policy takes place, subject to the condition that no other job is released or ends before $t_{\mathrm{tent}}$. Notice that $B(t)$ may be empty, and $t_{\mathrm{tent}} = \infty$ implies that the next action of the policy takes place when the next job is released or some job ends, whatever occurs first. Of course, the definition of $B(t)$ must respect potential release dates, precedence constraints, and the number of available machines. A policy is required to be *nonanticipatory*, meaning that the action of a policy at any time $t$ must depend only on the state of the system at time $t$ (together with the given input data, of course). The time instances at which a policy takes its actions are called *decision times.* Given an action of a policy at a decision time $t$, the next decision time is $t_{\mathrm{tent}}$, or the time of the next job completion, or the time when the next job is released, whatever occurs first. Depending on the action of the policy, the state at the next decision time is realized according to the (conditional) probability distributions of the jobs' processing times.

A given policy eventually yields a feasible $m$-machine schedule for each realization $p$ of the processing times. For a given policy, denoted by $\Pi$, let $S_j^{\Pi}(p)$ and $C_j^{\Pi}(p)$ denote the start and completion times, respectively, of job $j$ for a given realization $p$, and let $S_j^{\Pi}(P)$ and $C_j^{\Pi}(P)$ denote the associated random variables.

**Approximation.** It follows from simple examples that, in general, a scheduling policy cannot yield the optimal schedule for each possible realization of the processing times; see, e.g., [21]. Hence, our goal is to find a policy $\Pi$ which minimizes the objective, say $Z^{\Pi}(P)$, in expectation. But even under this mild notion of optimality, few special cases exist for which optimal scheduling policies are known to be efficiently computable. One example is the optimality of list scheduling according to SEPT (shortest expected processing time first) for the problem without precedence constraints or release dates, with unit weights, and with exponentially distributed processing times, $\mathrm{P}|\,p_j \sim \exp(\lambda_j)|\mathrm{E}\,[\sum C_j]$ [1, 24]. This result was extended by Kämpke [12] to the case where the weights $w_j$ are compliant with the expected processing times. In general, however, there exist examples which show that optimal policies can be rather complicated in the sense that they must indeed utilize the full information on the conditional distributions of the jobs' processing times; see, e.g., [22]. In this paper, we therefore concentrate on approximation algorithms. In stochastic scheduling, a scheduling policy $\Pi$ is said to be an *$\alpha$-approximation* if its expected performance $\mathrm{E}[Z^{\Pi}(P)]$ is within a factor of $\alpha$ of the expected performance $\mathrm{E}[Z^{\Pi^*}(P)]$ of an optimal (nonanticipatory) scheduling policy $\Pi^*$. The value $\alpha$ is called the *performance guarantee.*

**List scheduling policies.** There exist essentially three different classes of list scheduling policies, all of which have in common that there is a fixed priority list $L$ of jobs which determines the order in which the jobs are considered. We call a job $j$ *available* with respect to a partial schedule at time $t$ if all predecessors of $j$ are completed by $t$ and if $r_j \leqslant t$.

*Graham's list scheduling.* This is perhaps the most natural class of policies, often referred to as *the* list scheduling algorithm of Graham [7, 8]. Iterating over decision times, it greedily starts as many available jobs as possible, always in the order of the list $L$. It always holds that $t_{\text{tent}} = \infty$, and jobs are thus started only at release dates or upon completion of other jobs. If precedence constraints or release dates exist, it may happen that the order of start times of jobs differs from the order of the jobs in the priority list $L$; the jobs are scheduled "out of order" with respect to the priority list $L$. For the deterministic problem with *makespan* objective, $\mathrm{P}|prec|C_{\max}$, it is well known that Graham's list scheduling achieves a performance guarantee of $2 - 1/m$ for any priority list of the jobs [7]. This result straightforwardly extends to stochastic processing times and the expected makespan objective, $\mathrm{P}|\,prec|\mathrm{E}\,[C_{\max}]$ [3]. For the expected total weighted completion time, Graham's list scheduling in the WSEPT order[1] yields a constant-factor approximation for the problem without precedence constraints or release dates, $\mathrm{P}||\mathrm{E}\,[\sum w_j C_j]$ [15]. In the presence of release dates or precedence constraints, even in the deterministic setting, there are examples which show that the performance of Graham's algorithm can be arbitrarily bad. For an example with precedence constraints, see [18].

*Job-based list scheduling.* This is, in fact, the same list scheduling policy as before, only with the additional constraint that no job is started earlier than any of its predecessors in the priority list $L$. Hence, this policy preserves the order of the jobs in the priority list $L$ at the cost of deliberate idle times on the machines. For the deterministic problem $\mathrm{P}|r_j, prec|\sum w_j C_j$, the currently best known performance guarantee of 4 relies on (a slight variation of) job-based list scheduling, and the priority list is defined on the basis of an optimal solution to a linear programming relaxation [16].

[1]In the WSEPT order, jobs appear in nonincreasing order of the ratios $w_j/\mathrm{E}[P_j]$.

For the stochastic problem without precedence constraints, $\mathrm{P}|r_j|\mathrm{E}[\sum w_j C_j]$, job-based list scheduling yields a constant performance guarantee, too. This result is also based on a priority list that is defined on the basis of an optimal solution to an LP-relaxation [15].

*Delayed list scheduling.* As a matter of fact, approximation results for stochastic parallel machine scheduling were previously known only for problems without precedence constraints [23, 15]. In this paper, we close this gap, relying on yet another class of list scheduling policies which generalizes both Graham's and job-based list scheduling algorithms. It has been suggested in a paper by Chekuri et al. [5] to obtain a 5.828-approximation for the deterministic problem $\mathrm{P}|r_j, prec|\sum w_j C_j$. We consider the analogous stochastic variant of this algorithm. The basic idea is to extend Graham's list scheduling in such a way that a job may be scheduled out of order only if a certain amount of deliberate idle time has accumulated before. Thus, in this algorithm we use values $t_{\mathrm{tent}} < \infty$, and jobs may be started at times different from release dates or completion times of other jobs. The algorithm is parametric on a parameter $\beta \geqslant 0$ that controls the tradeoff between the amount of out-of-order processing of jobs and the desire to adhere to the order of the given priority list $L$. For $\beta = 0$ and $\beta = \infty$ we get Graham's and job-based list scheduling algorithms, respectively. The algorithm will be described in more detail in section 2.

**Contribution of this paper.** We derive the first constant performance guarantees for stochastic parallel machine scheduling with precedence constraints. The results are derived by borrowing heavily from two previous approaches. On the one hand, we use (an appropriate adaptation of) the delayed list scheduling algorithm of Chekuri et al. [5]. On the other hand, the priority list is derived from an optimal solution for (a generalized version of) the LP-relaxation by Möhring, Schulz, and Uetz [15]. It seems, however, that only this combination of the previous techniques is capable of yielding the desired approximation results.

Table 1 gives an overview of performance guarantees for stochastic parallel machine scheduling problems with the total weighted completion time objective. The last column indicates which of the results are proved in [15]; the asterisk [$*$] indicates that the results are derived in this paper. The term $\Delta$ denotes some common upper bound on the values $\mathrm{Var}[P_j]/(\mathrm{E}[P_j])^2$ for all jobs $j \in V$. In other words, $\sqrt{\Delta}$ is a common upper bound on the coefficient of variation $\mathrm{CV}[P_j] = \sqrt{\mathrm{Var}[P_j]}/\mathrm{E}[P_j]$ for all processing time distributions $P_j$, $j \in V$. Moreover, the number of machines is denoted by $m$, and $\beta$ is the nonnegative parameter used to control the delayed list scheduling algorithm. The third column shows the respective performance bounds for processing time distributions where the coefficient of variation is bounded by 1, which is the case for exponential, uniform, or Erlang distributions, to name a few.

**Relations to online optimization and other models.** Compared to the model described above, *online optimization* is another way of coping with the fact that the future is uncertain. We refer to Fiat and Woeginger [6] for details about online optimization. There is, however, a significant difference between the underlying paradigms of the above described analysis and the usual competitive analysis that prevails in online optimization. First, competitive analysis is based upon the a posteriori comparison, *"What was achieved under uncertainty about the future, and what could have been achieved if the future would not have been uncertain?"* This is expressed by the fact that the *adversary* is generally an oracle that knows the optimal solution. In contrast, stochastic scheduling addresses the a priori question, *"What is*

TABLE 1
*Performance bounds for stochastic scheduling problems. Asterisks* [$*$] *mark results of this paper.*

| Scheduling model | Performance guarantee | | |
|---|---|---|---|
| | Arbitrary $P_j$ | $\mathrm{CV}[P_j] \leqslant 1$ | |
| $1\|prec\|\mathrm{E}\left[\sum w_j C_j\right]$ | $2$ | $2$ | [15] |
| $1\|r_j, prec\|\mathrm{E}\left[\sum w_j C_j\right]$ | $3$ | $3$ | [15] |
| $\mathrm{P}\| \|\mathrm{E}\left[\sum w_j C_j\right]$ | $1 + \frac{(m-1)(\Delta+1)}{2m}$ | $2 - \frac{1}{m}$ | [15] |
| $\mathrm{P}\|r_j\|\mathrm{E}\left[\sum w_j C_j\right]$ | $3 - \frac{1}{m} + \max\{1, \frac{m-1}{m}\Delta\}$ | $4 - \frac{1}{m}$ | [15] |
| $\mathrm{P}\|in\text{-}forest\|\mathrm{E}\left[\sum w_j C_j\right]$ | $2 - \frac{1}{m} + \max\{1, \frac{m-1}{m}\Delta\}$ | $3 - \frac{1}{m}$ | [$*$] |
| $\mathrm{P}\|prec\|\mathrm{E}\left[\sum w_j C_j\right]$ | $(1+\beta)\big(1 + \frac{m-1}{m\,\beta} + \max\{1, \frac{m-1}{m}\Delta\}\big)$ | $3 + 2\sqrt{2} - \frac{1+\sqrt{2}}{m}$ | [$*$] |
| $\mathrm{P}\|r_j, prec\|\mathrm{E}\left[\sum w_j C_j\right]$ | $(1+\beta)\big(1 + \frac{1}{\beta} + \max\{1, \frac{m-1}{m}\Delta\}\big)$ | $3 + 2\sqrt{2}$ | [$*$] |

*the best that can be achieved under the given uncertainty about the future?"* Here, the underlying adversary is much weaker: the adversary must not anticipate future information, just like the policy itself. Second, in competitive analysis the adversary is allowed to determine, to a certain extent, the input distribution. This is not the case in the stochastic model considered here, since the input distributions are considered exogenous. It is interesting to note that two generalized online frameworks were suggested by Koutsoupias and Papadimitriou [13]. They restrict the adversary's power in two ways: its ability to choose an input distribution, and its ability to find an optimal solution. To some extent, the stochastic scheduling model incorporates both ideas, too. We refer to [13] for details and to [21] for a brief discussion. Another type of analysis for stochastic models has been proposed recently by Scharbrodt, Schickinger, and Steger [17]. They analyze the *expected competitive ratio* $\mathrm{E}[Z^{\Pi}(P)/Z^{\mathrm{OPT}}(P)]$, where $Z^{\mathrm{OPT}}(p)$ is the optimal solution value for a realization $p$. In this type of analysis the adversary is again an oracle that knows the optimal solution. We refer to [17] for a more detailed discussion of the benefits of their approach in comparison to the approach of this paper.

## 2. List scheduling with deliberate idle times.

We start with a few preliminaries that will be used later in the analysis. First, recall that we refer to a job $j$ *available* with respect to a partial schedule at time $t$ if all predecessors of $j$ are completed by $t$ and if $r_j \leqslant t$.

ASSUMPTION 2.1. *For any instance of* $\mathrm{P}|r_j, prec|\,\gamma$, *assume that* $r_j \geqslant r_i$ *whenever job* $i$ *is a predecessor of job* $j$ *in the precedence constraints.*

(Here, $\gamma$ is used to denote an arbitrary objective function.) Obviously, Assumption 2.1 can be made without loss of generality. Additionally, we use the following definitions.

DEFINITION 2.2 (critical predecessor). *Let some realization* $p$ *of the processing times and a feasible schedule be given. For any job* $j$, *a critical predecessor of* $j$ *is a predecessor* $i$ *of* $j$ (*with respect to the precedence constraints*) *with* $C_i > r_j$ *and* $C_i$ *maximal among all predecessors.*

DEFINITION 2.3 (critical chain). *Let some realization* $p$ *of the processing times and a feasible schedule be given. For a given job* $j$, *a critical chain for job* $j$ *and its length* $\ell_j(p)$ *is defined backwards recursively: If* $j$ *has no critical predecessor,* $j$ *is the only job in the critical chain, and* $\ell_j(p) = r_j + p_j$. *Otherwise,* $\ell_j(p) = p_j + \ell_k(p)$, *where job* $k$ *is a critical predecessor of job* $j$.

Definition 2.3 is illustrated in Figure 1. Notice that the critical chain as well

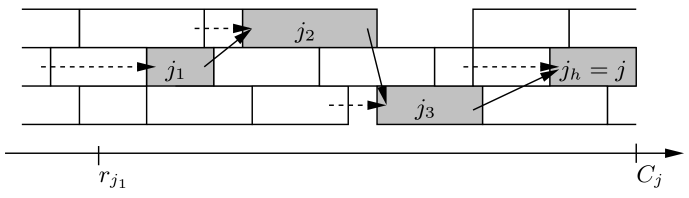


FIG. 1. *Example of a critical chain for job $j$. Its length is $\ell_j(p) = r_{j_1} + \sum_{i=1}^{h} p_{j_i}$ .*

as its length $\ell_j(p)$ depend on both the realization of the processing times $p$ and the underlying schedule. Moreover, since a critical predecessor is not necessarily unique, the critical chain and its length also depend on a tie-breaking rule for choosing critical predecessors. This is not relevant for our analysis, but in order to make the above definition unique, let us suppose that some arbitrary but fixed tie-breaking rule is used. Notice further that the first job $j_1$ of a critical chain is available at its release date $r_{j_1}$. This follows directly from the definition.

Like Graham's list scheduling, the algorithm we use iterates over decision times until all jobs have been scheduled. Assume a priority list $L$ is given. As with job-based list scheduling, the algorithm strives to schedule the jobs in the order of the list $L$ by leaving deliberate idle times. But if the accumulating deliberate idle time exceeds a certain threshold, the algorithm "panics" and schedules the first available job from the list. The algorithm is parametric on a parameter $\beta \geqslant 0$ that controls the tradeoff between the amount of out-of-order processing of jobs and the desire to adhere to the order of the given priority list $L$. At each stage of the algorithm, the sublist of $L$ containing all jobs that are not yet scheduled is referred to as the *residual list.* The following is a direct adaptation of the algorithm introduced by Chekuri et al. [5].

ALGORITHM CMNS (CHEKURI–MOTWANI–NATARAJAN–STEIN).[2] Whenever a machine is idle and the first job in the residual list is available, the job is scheduled. Otherwise, if the first job is not available, the first available job $j$ in the residual list (if any) is *deliberately delayed.* If $j$ was deliberately delayed for an accumulated time of $\beta\mathrm{E}[P_j]$, it is scheduled *out of order.*

We emphasize that *deliberate idle time* accumulates $m'$ times faster when a job is deliberately delayed while $m'$ machines are idle. For the purpose of analyzing the performance of Algorithm CMNS, any job $j$ gets *charged* the amount of deliberate idle time that accumulates during time intervals when $j$ is deliberately delayed. An alternative interpretation is the following: whenever a job $j$ is deliberately delayed, the tentative next decision time $t_{\text{tent}}$ is that point in time where the accumulated deliberate idle time charged to job $j$ would equal $\beta\mathrm{E}[P_j]$.

Analogous to [5], we introduce some additional notation. For a given job $j$, denote by $B_j$ and $A_j$ the sets of jobs that come before and after job $j$ in the priority list $L$, respectively; by convention, $B_j$ also includes job $j$. For the remaining definitions, we consider a fixed realization $p$ of the processing times and the resulting schedule constructed by Algorithm CMNS. Then, $r_j(p) \geqslant r_j$ denotes the earliest point in time when job $j$ becomes available; let $j_1, j_2, \ldots, j_h = j$ be the critical chain for job $j$ and

[2]The only difference between CMNS and the algorithm presented in [5] is the use of the threshold $\beta\mathrm{E}[P_j]$ instead of $\beta p_j$. CMNS coincides with the classical list scheduling algorithm of Graham [7] if we choose $\beta = 0$ and coincides with the job-based list scheduling algorithm if we choose $\beta = \infty$.

define $B_j(p) := B_j \setminus \{j_1, \dots, j_h\}$. That is, the set $B_j(p)$ contains all jobs that come before job $j$ in the priority list $L$, except for those which belong to the critical chain. Moreover, let $O_j(p) \subseteq A_j$ be the jobs in $A_j$ that are started out of order, that is, before $j$.

The following observation is the analogue to the results for the deterministic setting by Chekuri et al. [5, Fact 4.6, Lemma 4.7].

*Observation* 2.4. For any realization $p$ of the processing times and any job $j$,

(i) job $j$ is charged no more than $\beta\,\mathrm{E}[P_j]$ deliberate idle time;

(ii) the deliberate idle time in $[r_j(p), S_j(p)[$ is charged only to jobs in $B_j$;

(iii) there is no uncharged deliberate idle time.

*Proof.* Part (i) follows by construction of the algorithm and part (iii) by definition of deliberate idle time. Finally, for (ii), observe that no job from $A_j$ is the first available job from the residual list in the time interval $[r_j(p), S_j(p)[$, since job $j$ is available from $r_j(p)$ on, and $j$ has higher priority than any job in $A_j$. ∎

The following analysis of Algorithm CMNS closely resembles the analysis performed in [5] for the deterministic case. We first derive an upper bound on the completion time of any job for a fixed realization $p$.

LEMMA 2.5. *Consider the schedule constructed by Algorithm CMNS for any $\beta \geqslant 0$, any realization $p$ of the processing times, and any priority list $L$ which is a linear extension of the precedence constraints. Let $C_j(p)$ denote the resulting completion time of any job $j$, and let $\ell_j(p)$ denote the length of the critical chain for job $j$. Then*

$$C_j(p) \;\leqslant\; \frac{m-1}{m}\,\ell_j(p) + \frac{1}{m}\,r_j + \frac{1}{m}\left(\sum_{i\in B_j}\bigl(p_i + \beta\,\mathrm{E}[P_i]\bigr) \;+\; \sum_{i\in O_j(p)} p_i\right). \tag{2.1}$$

*Proof.* The basic idea resembles Graham's analysis for the makespan objective [7]. Consider the critical chain for job $j$ with total length $\ell_j(p)$, consisting of jobs $j_1, j_2, \dots, j_h = j$. Now partition the interval $[r_{j_1}, C_j(p)[$ into time intervals, where some job from the critical chain is in process, and the remaining time intervals. The latter are exactly $[r_i(p), S_i(p)[$, $i = j_1, \dots, j_h$. (Recall that $r_{j_1} = r_{j_1}(p)$ due to the definition of the critical chain.) By definition,

$$C_j(p) \;=\; \ell_j(p) + \sum_{i=j_1}^{j_h}\bigl(S_i(p) - r_i(p)\bigr). \tag{2.2}$$

To bound the total length of the intervals $[r_i(p), S_i(p)[$, $i = j_1, \dots, j_h$, observe that in each of these intervals there is no idle time except (possibly) deliberate idle time, since job $i$ is available in $[r_i(p), S_i(p)[$. Hence, the total processing in these intervals can be partitioned into three categories as follows:

– processing of jobs from $B_j$ which do not belong to the critical chain, i.e., jobs in $B_j(p)$;

– deliberate idle time;

– processing of jobs from $A_j$ which are scheduled out of order, i.e., jobs in $O_j(p)$.

Due to Observation 2.4(ii), all deliberate idle time in the interval $[r_i(p), S_i(p)[$ is charged only to jobs in $B_i$, $i = j_1, \dots, j_h$. Since the priority list $L$ is a linear extension of the precedence constraints, we have $B_{j_1} \subset B_{j_2} \subset \cdots \subset B_{j_h} = B_j$. Hence, all deliberate idle time in the intervals $[r_i(p), S_i(p)[$, $i = j_1, \dots, j_h$, is charged only to jobs in $B_j$. Since there is no uncharged deliberate idle time (Observation 2.4(iii)), and since each job $i \in B_j$ gets charged no more than $\beta\,\mathrm{E}[P_i]$ idle time (Observation 2.4(i)), the total amount of deliberate idle time in the intervals $[r_i(p), S_i(p)[$, $i = j_1, \dots, j_h$, is bounded from above by $\beta \sum_{i\in B_j} \mathrm{E}[P_i]$. This yields

$$\sum_{i=j_1}^{j_h} \big(S_i(p) - r_i(p)\big) \leqslant \frac{1}{m}\left(\sum_{i\in B_j(p)} p_i + \sum_{i\in B_j} \beta\,\mathrm{E}[P_i] + \sum_{i\in O_j(p)} p_i\right). \tag{2.3}$$

Finally, due to Assumption 2.1 we have $r_{j_1} \leqslant r_j$; thus

$$\sum_{i\in B_j(p)} p_i \leqslant \sum_{i\in B_j} p_i - \big(\ell_j(p) - r_j\big). \tag{2.4}$$

Now put (2.4) into (2.3), and then (2.3) into (2.2), and the claim follows. □

Before we take expectations in (2.1), we concentrate on the term $\sum_{i\in O_j(p)} p_i$. The following lemma shows that the expected total processing time of the jobs in $O_j(p)$—the jobs that are scheduled out of order with respect to $j$ (and $p$)—is independent of their actual processing times.

LEMMA 2.6. *We have*

$$\mathrm{E}\left[\sum_{i\in O_j(P)} P_i\right] = \mathrm{E}\left[\sum_{i\in O_j(P)} \mathrm{E}[P_i]\right].$$

*Proof.* We can write $\sum_{i\in O_j(P)} P_i$ equivalently as $\sum_{i\in A_j} \delta_i(P)\,P_i$, where $\delta_i(P)$ is a binary random variable which is 1 if and only if $i \in O_j(p)$. Linearity of expectation yields

$$\mathrm{E}\left[\sum_{i\in O_j(P)} P_i\right] = \mathrm{E}\left[\sum_{i\in A_j} \delta_i(P)\,P_i\right] = \sum_{i\in A_j} \mathrm{E}[\delta_i(P)\,P_i].$$

Notice that $\delta_i(P)$ is dependent on $\beta\mathrm{E}[P_i]$ but stochastically independent of $P_i$, as the decision to process job $i$ out of order is made before it is actually processed. (Here we require that the processing times be stochastically independent and that policies be nonanticipatory.) Hence,

$$\sum_{i\in A_j} \mathrm{E}[\delta_i(P)P_i] = \sum_{i\in A_j} \mathrm{E}[\delta_i(P)]\mathrm{E}[P_i] = \sum_{i\in A_j} \mathrm{E}\big[\delta_i(P)\mathrm{E}[P_i]\big] = \mathrm{E}\left[\sum_{i\in O_j(P)} \mathrm{E}[P_i]\right].$$

This concludes the proof. □

The following lemma bounds the expected amount of processing time of jobs from $A_j$ which are scheduled out of order in terms of the expected length of the critical chain for job $j$; compare to [5, Lemma 4.8].

LEMMA 2.7. *We have*

$$\frac{1}{m}\,\mathrm{E}\left[\sum_{i\in O_j(P)} \mathrm{E}[P_i]\right] \leqslant \frac{1}{\beta}\,\mathrm{E}[\ell_j(P)].$$

*Proof.* Consider a fixed realization $p$ of the processing times. If some job $i \in A_j$ is scheduled out of order, $i$ gets charged exactly $\beta\,\mathrm{E}[P_i]$ deliberate idle time. Hence, the total amount of deliberate idle time in $[0, S_j(p)[$ that is charged to jobs in $O_j(p)$ is $\beta \sum_{i \in O_j(p)} \mathrm{E}[P_i]$. Now consider the critical chain $j_1, \ldots, j_h = j$ for job $j$ with total length $\ell_j(p)$. From the proof of Lemma 2.5, we know that all deliberate idle time in the intervals $[r_i(p), S_i(p)[$, $i = j_1, \ldots, j_h$, is charged only to jobs in $B_j$. In other words, all deliberate idle time in $[0, S_j(p)[$ that is charged to jobs in $A_j$ lies in the complementary intervals $[0, r_{j_1}[$ and $[S_i(p), C_i(p)[$, $i = j_1, \ldots, j_{h-1}$. (Recall that $r_{j_1} = r_{j_1}(p)$ due to the definition of a critical chain.) The total length of these intervals is exactly $\ell_j(p) - p_j$. Hence, the total amount of deliberate idle time in $[0, S_j(p)[$ that is charged to jobs in $A_j$ is at most $m\,(\ell_j(p) - p_j) \leqslant m\,\ell_j(p)$. Hence, we obtain $\beta \sum_{i \in O_j(p)} \mathrm{E}[P_i] \leqslant m\,\ell_j(p)$ for any realization $p$ of the processing times. Taking expectations yields the claimed result. ☐

Finally, we obtain an upper bound on the expected completion time of any job under Algorithm CMNS; compare to [5, Theorem 4.9].

THEOREM 2.8. *For any instance of a stochastic scheduling problem* $\mathrm{P}|r_j, prec|\,\gamma$ *and any priority list L, which is a linear extension of the precedence constraints, the expected completion time of any job j under Algorithm CMNS* (*with parameter* $\beta \geqslant 0$) *fulfills*

$$\mathrm{E}[C_j(P)] \;\leqslant\; \left(\frac{m-1}{m} + \frac{1}{\beta}\right) \mathrm{E}[\ell_j(P)] \;+\; \frac{1+\beta}{m} \sum_{i \in B_j} \mathrm{E}[P_i] \;+\; \frac{1}{m}\, r_j\,. \tag{2.5}$$

(Again, $\gamma$ is used to denote an arbitrary objective function.)

*Proof.* Taking expectations in (2.1) together with Lemma 2.6 yields

$$\mathrm{E}[C_j(P)] \leqslant \frac{m-1}{m}\mathrm{E}[\ell_j(P)] + \frac{1}{m}\left( r_j \;+\; (1+\beta) \sum_{i \in B_j} \mathrm{E}[P_i] \;+\; \mathrm{E}\left[ \sum_{i \in O_j(P)} \mathrm{E}[P_i] \right] \right).$$

Plugging in the inequality from Lemma 2.7 gives the desired result. ☐

## 3. LP-relaxation.

To obtain a priority list $L$ as input for Algorithm CMNS, and to obtain a lower bound on the optimum, Chekuri et al. [5] use a single machine relaxation. This approach does not help in the stochastic setting, since the single machine problem does not necessarily provide a lower bound for the parallel machine problem; see [15, Ex. 4.1] for an example. Instead, we use LP-relaxations, which extend those used by Möhring, Schulz, and Uetz [15], adding inequalities which represent the precedence constraints. First, define $f : 2^V \to \mathbb{R}$ by

$$f(W) \;:=\; \frac{1}{2m}\left( \left( \sum_{j \in W} \mathrm{E}[P_j] \right)^2 + \sum_{j \in W} \mathrm{E}[P_j]^2 \right) - \frac{(m-1)(\Delta - 1)}{2m} \left( \sum_{j \in W} \mathrm{E}[P_j]^2 \right) \tag{3.1}$$

for $W \subseteq V$. Here, $\Delta \geqslant 0$ is a common upper bound on $\mathrm{Var}[P_j]/\mathrm{E}[P_j]^2$ for all jobs $j \in V$, where $\mathrm{Var}[P_j] = \mathrm{E}[P_j^2] - \mathrm{E}[P_j]^2$ is the variance of $P_j$. In other words, the *coefficient of variation*

$$\mathrm{CV}[P_j] \;:=\; \frac{\sqrt{\mathrm{Var}[P_j]}}{\mathrm{E}[P_j]}$$

of the distributions $P_j$ is bounded by $\sqrt{\Delta}$ for all $j \in V$. The following *load inequalities* are crucial for the derivation of our results.

THEOREM 3.1 (see [15, Cor. 3.1]). *If* $\mathrm{CV}[P_j] \leqslant \sqrt{\Delta}$ *for all* $P_j$ *and some* $\Delta \geqslant 0$, *the load inequalities*

$$\sum_{j\in W} \mathrm{E}[P_j]\, \mathrm{E}[C_j^{\Pi}(P)] \;\geqslant\; f(W) \tag{3.2}$$

*are valid for all* $W \subseteq V$ *and any nonanticipatory scheduling policy* $\Pi$.

In fact, as mentioned in [15], assuming that an upper bound exists on the coefficients of variation of the processing time distributions $P_j$ can be a reasonable assumption for many scheduling problems. For instance, assume that job processing times follow so-called *NBUE distributions*.

DEFINITION 3.2 (NBUE). *A nonnegative random variable* $X$ *is "new better than used in expectation" (NBUE) if* $\mathrm{E}\,[X - t | X > t] \leqslant \mathrm{E}\,[X]$ *for all* $t \geqslant 0$.

Here, $\mathrm{E}\,[X - t | X > t]$ is the conditional expectation of $X - t$ under the assumption that $X > t$. Roughly speaking, when processing times are NBUE, on average it is not disadvantageous to process a job. Examples for NBUE distributions are, among others, exponential, uniform, and Erlang distributions. A result of Hall and Wellner [11] states that the coefficient of variation $\mathrm{CV}[X]$ of any NBUE distribution $X$ is bounded by 1. Hence, by choosing $\Delta = 1$ the second term of the right-hand side of (3.2) can be neglected for NBUE distributions, which leads to simplified performance guarantees in section 4.

Observe that under any scheduling policy $\Pi$ the trivial inequalities

$$\mathrm{E}[C_j^{\Pi}(P)] \;\geqslant\; \mathrm{E}[C_i^{\Pi}(P)] + \mathrm{E}[P_j], \qquad (i,j) \in A,$$

and

$$\mathrm{E}[C_j^{\Pi}(P)] \;\geqslant\; \mathrm{E}[P_j], \qquad j \in V,$$

are valid, since they even hold pointwise for any realization of the processing times. Due to Theorem 3.1, the following is thus an LP-relaxation for the problem $\mathrm{P}|r_j, prec|\mathrm{E}\,[\sum w_j\, C_j]$:

$$\begin{aligned}
\text{Minimize} \quad & \sum_{j\in V} w_j\, C_j^{\mathrm{LP}} & \\
\text{subject to} \quad & \sum_{j\in W} \mathrm{E}[P_j]\, C_j^{\mathrm{LP}} \geqslant f(W), & W \subseteq V,\\
& C_j^{\mathrm{LP}} \geqslant C_i^{\mathrm{LP}} + \mathrm{E}[P_j], & (i,j) \in A,\\
& C_j^{\mathrm{LP}} \geqslant \mathrm{E}[P_j], & j \in V,
\end{aligned}$$

where $f : 2^V \to \mathbb{R}$ is the set function defined in (3.1). It is known that the load inequalities $\sum_{j\in W} \mathrm{E}[P_j]\, C_j^{\mathrm{LP}} \geqslant f(W)$, $W \subseteq V$, can be separated in time $O(n \log n)$ [15, 21]. Hence, due to the fact that the remaining number of inequalities is polynomial in terms of $n$, this LP-relaxation can be solved in time polynomial in $n$ by the equivalence of separation and optimization [10]. The following technical lemma of Möhring, Schulz, and Uetz [15] is required later in the analysis.

LEMMA 3.3 (see [15, Lemma 4.2]). *Let* $C^{LP} \in \mathbb{R}^n$ *be any point that satisfies the first and the last set of inequalities from the LP-relaxation. Assuming* $C_1^{LP} \leqslant C_2^{LP} \leqslant \cdots \leqslant C_n^{LP}$, *we then have for all* $j \in V$

$$\frac{1}{m}\sum_{k=1}^{j} \mathrm{E}[P_k] \;\leqslant\; \left(1 + \max\left\{1, \frac{m-1}{m}\Delta\right\}\right) C_j^{LP}.$$

## 4. Results.

We are now ready to prove approximation results for stochastic machine scheduling problems with precedence constraints.

**General precedence constraints.** We consider the general problem with precedence constraints and release dates, $\mathrm{P}|r_j, prec|\mathrm{E}\left[\sum w_j\, C_j\right]$. From an optimal solution for the LP-relaxation, we define a priority list $L$ according to nondecreasing "LP completion times" $C_j^{\mathrm{LP}}$. It is perhaps interesting to note that inequalities $C_j^{\mathrm{LP}} \geqslant C_i^{\mathrm{LP}} + \mathrm{E}[P_j]$, $(i,j) \in A$, are required only to ensure that the order according to nondecreasing LP completion times $C_j^{\mathrm{LP}}$ is a linear extension of the precedence constraints. They are not required elsewhere in the analysis. Moreover, instead of the weaker inequalities $C_j^{\mathrm{LP}} \geqslant \mathrm{E}[P_j]$ we could use $C_j^{\mathrm{LP}} \geqslant r_j + E[P_j]$ as well, but this does not yield an improvement of our results.

Theorem 4.1. *Consider an instance of the stochastic machine scheduling problem* $\mathrm{P}|r_j, prec|\mathrm{E}\left[\sum w_j\, C_j\right]$ *with* $\mathrm{CV}[P_j] \leqslant \sqrt{\Delta}$ *for all processing times* $P_j$ *and some* $\Delta \geqslant 0$. *Let* $L$ *be a priority list according to an optimal solution* $C^{LP}$ *of the LP-relaxation. Then Algorithm CMNS* (*with parameter* $\beta > 0$) *is an* $\alpha$*-approximation with*

$$\alpha \;=\; (1+\beta)\left(1 + \frac{1}{\beta} + \max\left\{1, \frac{m-1}{m}\Delta\right\}\right).$$

*Proof.* Since $L$ is a linear extension of the precedence constraints, Theorem 2.8 yields

$$\mathrm{E}[C_j(P)] \;\leqslant\; \left(\frac{m-1}{m} + \frac{1}{\beta}\right)\mathrm{E}[\ell_j(P)] \;+\; \frac{1+\beta}{m}\sum_{i\in B_j}\mathrm{E}[P_i] \;+\; \frac{1}{m}\, r_j$$

for any job $j \in V$. (Recall that $B_j$ denotes the jobs that come before job $j$ in the priority list $L$.) Lemma 3.3 yields

$$\frac{1}{m}\sum_{i\in B_j}\mathrm{E}[P_i] \;\leqslant\; \left(1 \;+\; \max\left\{1, \frac{m-1}{m}\Delta\right\}\right) C_j^{\mathrm{LP}}$$

for all $j \in V$. Hence,

$$\sum_{j\in V} w_j\, \mathrm{E}[C_j(P)] \leqslant \left(\frac{m-1}{m} + \frac{1}{\beta}\right)\sum_{j\in V} w_j\, \mathrm{E}[\ell_j(P)] + (1+\beta)\left(1 \;+\; \max\left\{1, \frac{m-1}{m}\Delta\right\}\right)\sum_{j\in V} w_j\, C_j^{\mathrm{LP}} + \frac{1}{m}\sum_{j\in V} w_j\, r_j\,.$$

Now, for any job $j$ and any realization $p$ of the processing times, the length $\ell_j(p)$ of a critical chain for job $j$ is a lower bound for job $j$'s completion time, $\ell_j(p) \leqslant C_j(p)$. This is true by definition of a critical chain. Hence, the value $\mathrm{E}[\ell_j(P)]$ is a lower bound on the expected completion time $\mathrm{E}[C_j(P)]$ of any job $j$ for any scheduling policy. (Notice that the critical chain may be different for different realizations of the processing times, and thus the fact that $\mathrm{E}[\ell_j(P)] \leqslant \mathrm{E}[C_j(P)]$ cannot be derived from the precedence constraints in the LP-relaxation.) Thus, $\sum_{j\in V} w_j\, \mathrm{E}[\ell_j(P)]$ is a lower bound on the expected performance of an optimal scheduling policy. Moreover, both terms $\sum_{j\in V} w_j\, C_j^{\mathrm{LP}}$ and $\sum_{j\in V} w_j\, r_j$ are lower bounds on the expected performance of an optimal scheduling policy as well. This gives a performance bound of

$$\left(\frac{m-1}{m} + \frac{1}{\beta}\right) + (1+\beta)\left(1 \;+\; \max\left\{1, \frac{m-1}{m}\Delta\right\}\right) + \frac{1}{m}.$$

Rearranging the terms yields the desired result. ∎

Notice that Theorem 4.1 implies a performance bound of $3 + 2\sqrt{2} \approx 5.828$ if $\beta = 1/\sqrt{2}$ and if the jobs' processing times are distributed according to NBUE distributions (see Definition 3.2). This matches the performance guarantee achieved in [5] for the corresponding deterministic scheduling problem $\mathrm{P}|r_j, prec|\sum w_j\, C_j$. The performance bound in Theorem 4.1 can be slightly improved if release dates are absent.

THEOREM 4.2. *Consider an instance of the stochastic machine scheduling problem* $\mathrm{P}|prec|\mathrm{E}\left[\sum w_j\, C_j\right]$ *with* $\mathrm{CV}[P_j] \leqslant \sqrt{\Delta}$ *for all processing times* $P_j$ *and some* $\Delta \geqslant 0$. *Let* $L$ *be a priority list according to an optimal solution* $C^{LP}$ *of the LP-relaxation. Then Algorithm CMNS* (*with parameter* $\beta > 0$) *is an* $\alpha$*-approximation with*

$$\alpha \;=\; (1+\beta)\left(1 + \frac{m-1}{m\,\beta} + \max\left\{1, \frac{m-1}{m}\Delta\right\}\right).$$

The tighter bound follows from two modifications in the proof of Theorem 4.1. On the one hand, in the proof of Lemma 2.7, one can show that

$$\frac{1}{m}\,\mathrm{E}\left[\sum_{i \in O_j(P)} \mathrm{E}[P_i]\right] \;\leqslant\; \frac{m-1}{m\,\beta}\,\mathrm{E}[\ell_j(P)].$$

The reason is that there are only $m - 1$ machines available for the deliberate idle time that is charged to jobs which are scheduled out of order: Simultaneous to the deliberate idle time, at least one job from the critical chain $j_1, j_2, \ldots, j_h$ is in process. (This argument does not hold if release dates are present, since deliberate idle time could possibly accumulate before $r_{j_1}$.) On the other hand, it is immediate that the last term $(1/m)\,r_j$ on the right-hand side of (2.5) disappears. With these modifications, the claim follows exactly as in Theorem 4.1.

**In-forest precedence constraints.** Let us now turn to the special case of the problem denoted by $\mathrm{P}|in\text{-}forest|\mathrm{E}\left[\sum w_j\, C_j\right]$. In-forest precedence constraints are characterized by the fact that each job has at most one successor. Moreover, we assume that there are no release dates. For this problem, the results of the preceding section can be further improved.

We start with the following observation which is also contained in [5, Lemma 4.16]; we nevertheless give a short proof for the sake of completeness.

LEMMA 4.3. *Consider the schedule constructed by Graham's list scheduling for an arbitrary priority list L, which is a linear extension of the* (*in-forest*) *precedence constraints, and any realization p of the processing times. Then, in the interval* $[r_j(p), S_j(p)[$ *there is no processing of jobs in* $A_j$.

*Proof.* Suppose the claim is false and, among all jobs which violate it, let job $j$ be one that is scheduled earliest. Obviously, $S_j(p) > r_j(p)$; otherwise the claim is trivially true. In the interval $[r_j(p), S_j(p)[$ no job from $A_j$ is started, since $j$ is available from time $r_j(p)$ on. Hence, there must be some job $k \in A_j$ that has been started before $r_j(p)$ and that is still in process at $r_j(p)$. Thus $r_j(p) > 0$. Denote by $h$ the number of jobs that are started at time $r_j(p)$. All of these jobs $i$ have higher priority than $j$, and the fact that $j$ is the first job that violates the claim yields $r_i(p) = r_j(p)$. (At this point it is crucial that the priority list extends the precedence constraints.) In other words, for each of these jobs a critical predecessor ends at time $r_j(p)$ and, due to the fact that the precedence constraints form an in-forest, all of these predecessors are different. Hence, including $j$'s critical predecessor, $h+1$ different jobs end at time

$r_j(p)$, but only $h$ are started. This is a contradiction since job $j$ is available at time $r_j(p)$. □

LEMMA 4.4. *For any instance of the stochastic scheduling problem* $\mathrm{P}|$*in-forest*$|\,\gamma$ *and any priority list $L$, which is a linear extension of the precedence constraints, the expected completion time of any job $j$ under Graham's list scheduling fulfills*

$$\mathrm{E}[C_j(P)] \leqslant \frac{m-1}{m}\mathrm{E}[\ell_j(P)] + \frac{1}{m}\sum_{i\in B_j}\mathrm{E}[P_i]. \tag{4.1}$$

(Again, $\gamma$ is used to denote an arbitrary objective function.)

*Proof.* Consider any realization $p$ of the processing times. Given any job $j$, consider a critical chain for $j$, consisting of jobs $j_1, j_2, \dots, j_h = j$ and with total length $\ell_j(p)$. The time interval $[0, C_j(p)]$ can be partitioned into time intervals, where a job from a critical chain for $j$ is in process, and the remaining time intervals. Due to Lemma 4.3, in each time interval $[r_i(p), S_i(p)[$ there is no job from $A_i$ in process for all $i = j_1, \dots, j_h$. Moreover, there is no idle time on any of the machines in these time intervals (we consider Graham's list scheduling, and there are no release dates). Since $A_{j_1} \supset A_{j_2} \supset \cdots \supset A_{j_h} = A_j$, it follows that the only jobs processing in these time intervals are the jobs in $B_j$, or more precisely, in $B_j(p)$. In other words, the total processing time of jobs in these time intervals is at most $\sum_{i\in B_j} p_i - \ell_j(p)$. Hence,

$$C_j(p) \leqslant \frac{m-1}{m}\ell_j(p) + \frac{1}{m}\sum_{i\in B_j} p_i$$

for any realization $p$. Taking expectations, we see that the claim follows. □

THEOREM 4.5. *Consider an instance of* $\mathrm{P}|$*in-forest*$|\mathrm{E}[\sum w_j C_j]$ *with* $\mathrm{CV}[P_j] \leqslant \sqrt{\Delta}$ *for all processing times $P_j$ and some $\Delta \geqslant 0$. Let $L$ be a priority list according to an optimal solution $C^{LP}$ of the LP-relaxation. Then Graham's list scheduling is an $\alpha$-approximation with*

$$\alpha = 2 - \frac{1}{m} + \max\left\{1, \frac{m-1}{m}\Delta\right\}.$$

*Proof.* Graham's list scheduling coincides with Algorithm CMNS for $\beta = 0$. The proof is therefore exactly the same as that of Theorem 4.1, except that Lemma 4.4 is used instead of Theorem 2.8. □

For NBUE distributions (see Definition 3.2), Theorem 4.5 yields a performance guarantee of $3 - 1/m$.

**Single machine problems.** Theorem 4.2 implies a 2-approximation for the special case of a single machine: In this case the term $(m-1)/(m\,\beta)$ disappears, and we can choose $\beta = 0$ to obtain performance guarantee 2. (For $\beta = 0$, the algorithm corresponds to Graham's list scheduling.) This holds for arbitrarily distributed, independent processing times. In fact, this matches the best bound currently known in the deterministic setting; see Open Problem 9 in the collection of Schuurman and Woeginger [19].

## 5. Further remarks.

A scheduling policy defines a mapping of processing times to start times of jobs. This mapping has to be universally measurable in order to grant existence of the expected objective function value [14]. Without going into further detail we just mention that the scheduling policies discussed in this paper fulfill this requirement; refer to [21, Cor. 3.6.15] for further details.

We point out that, apart from the expected processing times of the jobs, a uniform upper bound on their coefficients of variation is the sole stochastic information required as input for the presented scheduling policy. Nevertheless, in our analysis we compare its performance to a lower bound on the performance of *any* nonanticipatory scheduling policy. This refers to the broadest possible sense of scheduling policies as defined by Möhring, Radermacher, and Weiss [14]. In particular, an optimal scheduling policy is therefore allowed to take advantage of the *complete* knowledge of the conditional distributions of the processing times, at any time.

Finally, we mention that our analysis indeed requires policies to be nonanticipatory, because the LP lower bound does not hold otherwise. This can be seen from the observation that an anticipatory "scheduling policy" could, for instance, compute an optimal schedule for any realization of the processing times. Theorem 3.1, however, is no longer valid in this case; see [21]. In other words, our analysis is based upon an adversary that is just as powerful as the scheduling policy itself. This constitutes a major difference compared to the rather "unfair" competitive analysis known from online optimization.

**Acknowledgment.** We thank the anonymous referee for many valuable comments and suggestions for improvements. In particular, the compact formulation of the algorithm in section 2 was kindly proposed by the referee.

## REFERENCES

[1] J. L. Bruno, P. J. Downey, and G. N. Frederickson, *Sequencing tasks with exponential service times to minimize the expected flowtime or makespan*, J. ACM, 28 (1981), pp. 100–113.

[2] S. Chakrabarti and S. Muthukrishnan, *Resource scheduling for parallel database and scientific applications*, in Proceedings of the 8th Annual ACM Symposium on Parallel Algorithms and Architectures, Padua, Italy, 1996, pp. 329–335.

[3] K. M. Chandy and P. F. Reynolds, *Scheduling partially ordered tasks with probabilistic execution times*, Oper. Syst. Rev., 9 (1975), pp. 169–177.

[4] C. Chekuri, R. Johnson, R. Motwani, B. Natarajan, B. Rau, and M. Schlansker, *An analysis of profile-driven instruction level parallel scheduling with application to super blocks*, in Proceedings of the 29th Annual IEEE/ACM International Symposium on Microarchitecture, Paris, France, 1996, pp. 58–69.

[5] C. Chekuri, R. Motwani, B. Natarajan, and C. Stein, *Approximation techniques for average completion time scheduling*, SIAM J. Comput., 31 (2001), pp. 146–166.

[6] A. Fiat and G. J. Woeginger, eds., *Online Algorithms: The State of the Art*, Lecture Notes in Comput. Sci. 1442, Springer, Berlin, 1998.

[7] R. L. Graham, *Bounds for certain multiprocessing anomalies*, Bell System Tech. J., 45 (1966), pp. 1563–1581.

[8] R. L. Graham, *Bounds on multiprocessing timing anomalies*, SIAM J. Appl. Math., 17 (1969), pp. 416–429.

[9] R. L. Graham, E. L. Lawler, J. K. Lenstra, and A. H. G. Rinnooy Kan, *Optimization and approximation in deterministic sequencing and scheduling: A survey*, Ann. Discrete Math., 5 (1979), pp. 287–326.

[10] M. Grötschel, L. Lovász, and A. Schrijver, *Geometric Algorithms and Combinatorial Optimization*, 2nd ed., Algorithms and Combinatorics 2, Springer, Berlin, 1993.

[11] W. J. Hall and J. A. Wellner, *Mean residual life*, in Proceedings of the International Symposium on Statistics and Related Topics, M. Csörgö, D. A. Dawson, J. N. K. Rao, and A. K. Md. E. Saleh, eds., Ottawa, ON, 1981, North-Holland, Amsterdam, pp. 169–184.

[12] T. Kämpke, *On the optimality of static priority policies in stochastic scheduling on parallel machines*, J. Appl. Probab., 24 (1987), pp. 430–448.

[13] E. Koutsoupias and C. H. Papadimitriou, *Beyond competitive analysis*, SIAM J. Comput., 30 (2000), pp. 300–317.

[14] R. H. Möhring, F. J. Radermacher, and G. Weiss, *Stochastic scheduling problems* I: *General strategies*, ZOR—Math. Methods Oper. Res., 28 (1984), pp. 193–260.

[15] R. H. Möhring, A. S. Schulz, and M. Uetz, *Approximation in stochastic scheduling: The power of LP-based priority policies*, J. ACM, 46 (1999), pp. 924–942.

[16] A. Munier, M. Queyranne, and A. S. Schulz, *Approximation bounds for a general class of precedence constrained parallel machine scheduling problems*, in Proceedings of the 6th International Conference on Integer Programming and Combinatorial Optimization, Houston, TX, 1998, R. Bixby, E. A. Boyd, and R. Z. Rios Mercado, eds., Lecture Notes in Comput. Sci. 1412, Springer, Berlin, pp. 367–382.

[17] M. Scharbrodt, T. Schickinger, and A. Steger, *A new average case analysis for completion time scheduling*, in Proceedings of the 34th ACM Symposium on Theory of Computing, Montréal, QB, 2002, pp. 170–178.

[18] A. S. Schulz, *Polytopes and Scheduling*, Ph.D. thesis, Institut für Mathematik, Technische Universität Berlin, Germany, 1996.

[19] P. Schuurman and G. J. Woeginger, *Polynomial time approximation algorithms for machine scheduling: Ten open problems*, J. Scheduling, 2 (1999), pp. 203–213.

[20] M. Skutella and M. Uetz, *Scheduling precedence-constrained jobs with stochastic processing times on parallel machines*, in Proceedings of the 12th Annual ACM-SIAM Symposium on Discrete Algorithms, SIAM, Philadelphia, ACM, New York, 2001, pp. 589–590.

[21] M. Uetz, *Algorithms for Deterministic and Stochastic Scheduling*, Cuvillier Verlag, Göttingen, Germany, 2002.

[22] M. Uetz, *When greediness fails: Examples from stochastic scheduling*, Oper. Res. Lett., 31 (2003), pp. 413–419.

[23] G. Weiss, *Turnpike optimality of Smith's rule in parallel machines stochastic scheduling*, Math. Oper. Res., 17 (1992), pp. 255–270.

[24] G. Weiss and M. Pinedo, *Scheduling tasks with exponential service times on non-identical processors to minimize various cost functions*, J. Appl. Probab., 17 (1980), pp. 187–202.

SIAM J. Comput.
Vol. 34, No. 4, pp. 803–824

# IMPROVED COMBINATORIAL ALGORITHMS FOR FACILITY LOCATION PROBLEMS[*]

MOSES CHARIKAR[†] AND SUDIPTO GUHA[‡]

**Abstract.** We present improved combinatorial approximation algorithms for the uncapacitated facility location problem. Two central ideas in most of our results are *cost scaling* and *greedy improvement*. We present a simple greedy local search algorithm which achieves an approximation ratio of $2.414+\epsilon$ in $\tilde{O}(n^2/\epsilon)$ time. This also yields a bicriteria approximation tradeoff of $(1+\gamma, 1+2/\gamma)$ for facility cost versus service cost which is better than previously known tradeoffs and close to the best possible. Combining greedy improvement and cost scaling with a recent primal-dual algorithm for facility location due to Jain and Vazirani, we get an approximation ratio of 1.853 in $\tilde{O}(n^3)$ time. This is very close to the approximation guarantee of the best known algorithm which is linear programming (LP)-based. Further, combined with the best known LP-based algorithm for facility location, we get a very slight improvement in the approximation factor for facility location, achieving 1.728. We also consider a variant of the capacitated facility location problem and present improved approximation algorithms for this.

**Key words.** approximation algorithms, facility location, local search, combinatorial optimization

**AMS subject classifications.** 68W25, 90B80, 90C27

**DOI.** 10.1137/S0097539701398594

## 1. Introduction.

In this paper, we present improved combinatorial algorithms for some facility location problems. Informally, in the (uncapacitated) facility location problem we are asked to select a set of facilities in a network to service a given set of customers, minimizing the sum of facility costs as well as the distance of the customers to the selected facilities. (The precise problem definition appears in section 1.2). This classical problem, first formulated in the early 1960s, has been studied extensively in the operations research and computer science communities and has recently received a lot of attention (see [13, 33, 16, 9, 10, 11, 23]).

The facility location problem is NP-hard, and recent work has focused on obtaining approximation algorithms for the problem. The version where the assignment costs do not form a metric can be shown to be as hard to approximate as the set-cover problem. Shmoys, Tardos, and Aardal [33] gave the first constant factor approximation algorithm for metric facility location, where the assignment costs are based on a metric distance function.[1] This was subsequently improved by Guha and Khuller [16], Chudak [9], and Chudak and Shmoys [10], who achieved the currently best known approximation ratio of $1 + 2/e \approx 1.736$. All of these algorithms are based on solving

[*]Received by the editors November 25, 2001; accepted for publication (in revised form) August 28, 2004; published electronically April 29, 2005. This work was done while both authors were at Stanford University, Stanford, CA, and their research was supported by the Pierre and Christine Lamond Fellowship and the IBM Cooperative Fellowship, respectively, along with NSF grant IIS-9811904 and NSF award CCR-9357849, with matching funds from IBM, Mitsubishi, Schlumberger Foundation, Shell Foundation, and Xerox Corporation.

http://www.siam.org/journals/sicomp/34-4/39859.html

[†]Department of Computer Science, Princeton University, 35 Olden Street, Princeton, NJ 08544 (moses@cs.princeton.edu).

[‡]Department of Computer Information Sciences, University of Pennsylvania, 3330 Walnut Street, Philadelphia, PA 19104 (sudipto@cis.upenn.edu).

[1]Henceforth in this paper we will drop the term “metric” for brevity; all results referred to here assume a metric distance function.

linear programming (LP) relaxations of the facility location problem and rounding the solution obtained. Korupolu, Plaxton, and Rajaraman [23] analyzed a well-known local search heuristic and showed that it achieves an approximation guarantee of $(5+\epsilon)$. However, the algorithm has a fairly high running time of $O(n^4 \log n/\epsilon)$.

The facility location problem has also received a lot of attention due to its connection with the $k$-median problem. Lin and Vitter [25] showed that their filtering technique [26] can be used to round a fractional solution to the LP relaxation of the (metric) $k$-median problem, obtaining an integral solution of cost $2(1+\frac{1}{\epsilon})$ times the fractional solution while using $(1+\epsilon)k$ medians (facilities). Their algorithm is based on a technique called *filtering*, which was also used in [33], and has since found numerous other applications.

Jain and Vazirani [19] gave primal-dual algorithms for the facility location and $k$-median problems, achieving approximation ratios of 3 and 6 for the two problems. The running time of their algorithm for facility location is $O(n^2 \log n)$. The running time of the result can be made linear with a loss in the approximation factor using results in [18] if the distance function is given as an oracle. For results regarding the $k$-median problem, see [21, 25, 35, 38, 37, 4, 5, 6, 1, 8, 7, 30, 2]. See section 7 for work subsequent to this paper.

**1.1. Our results.** We present improved combinatorial algorithms for the facility location problem. The algorithms combine numerous techniques based on two central ideas. The first idea is that of *cost scaling*, i.e., to scale the costs of facilities relative to the assignment costs. The scaling technique exploits asymmetric approximation guarantees for the facility cost and the service cost. The idea is to apply the algorithm to the *scaled* instance and then scale back to get a solution for the original instance. On several occasions, this technique alone improves the approximation ratio significantly. The second idea used is *greedy local improvement.* If either the service cost or the facility cost is very high, greedy local improvement decreases the total cost by balancing the two. We show that greedy local improvement by itself yields a very good approximation for facility location in $\tilde{O}(n^2)$ time.

We first present a simple local search algorithm for facility location. This differs from (and is more general than) the heuristic proposed by Kuehn and Hamburger [24] and analyzed by Korupolu, Plaxton, and Rajaraman [23]. Despite the seemingly more complex local search step, each step can still be performed in $O(n)$ time. We are not aware of any prior work involving the proposed local search algorithm. We show that the (randomized) local search algorithm together with scaling yields an approximation ratio of $2.414+\epsilon$ in expected time $O(n^2(\log n + \frac{1}{\epsilon}))$ (with a multiplicative $\log n$ factor in the running time for a high probability result). This improves significantly on the local search algorithms considered by Korupolu, Plaxton, and Rajaraman, both in terms of running time and approximation guarantee. It also improves on the approximation guarantee of Jain and Vazirani, while still running in $\tilde{O}(n^2)$ time. We can also use the local search algorithm with scaling to obtain a bicriteria approximation for the facility cost and the service cost. We get a $(1+\gamma, 1+2/\gamma)$ tradeoff for facility cost versus service cost (within factors of $(1+\epsilon)$ for arbitrarily small $\epsilon$). Moreover, this holds even when we compare it with the cost of an arbitrary feasible solution to the facility location LP. Thus this yields a better tradeoff than that obtained by the *filtering* technique of Lin and Vitter [25] and the tradeoffs in [33, 23]. This gives bicriteria approximations for budgeted versions of facility location where the objective is to minimize either the facility cost or service cost subject to a budget constraint on the other. Further, our tradeoff is close to the best possible. We show that it is not possible to obtain a tradeoff better than $(1+\gamma, 1+1/\gamma)$.

Using both the local search algorithm and the primal-dual algorithm of [19] results in a $2\frac{1}{3}$ approximation for facility location in $\tilde{O}(n^2)$ time. We prove that a modified greedy improvement heuristic as in [16], along with the primal-dual algorithm in [19], together with scaling, yields an approximation ratio of 1.853 in $\tilde{O}(n^3)$ time. Interestingly, applying both this combinatorial algorithm (with appropriate scaling) and the LP-based algorithm of [9, 10] and taking the better of the two gives an approximation ratio of 1.728, marginally improving the best known approximation result for facility location. We construct an example to show that the dual constructed by the primal-dual facility location algorithm can be a factor $3-\epsilon$ away from the optimal solution. This shows that an analysis that uses only this dual as a lower bound cannot achieve an approximation ratio better than $3-\epsilon$.

Jain and Vazirani [19] show how a version of facility location with capacities (where multiple facilities are allowed at the same location) can be solved by reducing it to uncapacitated facility location. They obtain a 4 approximation for the capacitated problem in $\tilde{O}(n^2)$ time. Using their idea, from a $\rho$ approximation algorithm for the uncapacitated case, one can obtain an approximation ratio of $2\rho$ for the capacitated case. This was also observed by Shmoys [31]. This fact therefore implies approximation ratios of 3.7 in $\tilde{O}(n^3)$ time and 3.46 using LP-based techniques for the capacitated problem.

**1.2. Problem definition.** The *uncapacitated facility location problem* is defined as follows: Given a graph with an edge metric $c$ and cost $f_i$ of opening a center (or facility) at node $i$, select a subset of facilities to open so as to minimize the cost of opening the selected facilities plus the cost of assigning each node to its closest open facility. The cost of assigning node $j$ to facility $i$ is $d_j c_{ij}$, where $c_{ij}$ denotes the distance between $i$ and $j$. The constant $d_j$ is referred to as the demand of the node $j$.

Due to the metric property, this problem is also referred to as the metric uncapacitated facility location problem. In this paper we will refer to this problem as the facility location problem. We will denote the total cost of the facilities in a solution as the facility cost and the rest as the service cost. These costs will be denoted by $F$ and $C$, subscripted appropriately to define the context. For the LP relaxations of the problem and its dual, see section 4.

The *k-median problem* is defined as follows: given $n$ points in a metric space, we must select $k$ of these to be centers (facilities) and then assign each input point $j$ to the selected center that is closest to it. If location $j$ is assigned to a center $i$, we incur a cost $d_j c_{ij}$. The goal is to select the $k$ centers so as to minimize the sum of the assignment costs.

We will mostly present proofs assuming unit demands; however, since the arguments will be on a node by node basis, the results will extend to arbitrary demands as well.

## 2. Facility location and local search.

In this section we describe and analyze a simple greedy local search algorithm for facility location.

Suppose $F$ is the facility cost and $C$ is the service cost of a solution. The objective of the algorithm is to minimize the cost of the solution $F+C$. The algorithm starts from an initial solution and repeatedly attempts to improve its current solution by performing local search operations.

The initial solution is chosen as follows. The facilities are sorted in increasing order of facility cost. Let $F_i$ be the total facility cost and let $C_i$ be the total service cost for the solution consisting of the first $i$ facilities in this order. We compute the

$F_i$ and $C_i$ values for all $i$ and choose the solution that minimizes $F_i + C_i$. Lemma 2.1 bounds the cost of the initial solution in terms of the cost of an arbitrary solution $SOL$, and Lemma 2.2 shows that the initial solution can be computed in $O(n^2)$ time.

Let $\mathcal{F}$ be the set of facilities in the current solution. Consider a facility $i$. We will try to improve the current solution by incorporating $i$ and possibly removing some centers from $\mathcal{F}$. (Note that it is possible that $i \in \mathcal{F}$. In fact, this is required for reasons that will be made clear later.) Some nodes $j$ may be closer to $i$ than their currently assigned facility in $\mathcal{F}$. All such nodes are reassigned to $i$. Additionally, some centers in $\mathcal{F}$ are removed from the solution. If we remove a center $i' \in \mathcal{F}$, then all nodes $j$ that were connected to $i'$ are now connected to $i$. Note that the total change in cost depends on which nodes we choose to connect to $i$ and which facilities we choose to remove from the existing solution. The *gain* associated with $i$ (referred to as gain($i$)) is the largest possible decrease in $F + C$ as a result of this operation. If $F + C$ only increases as a result of adding facility $i$, gain($i$) is said to be 0. Lemma 2.3 guarantees that gain($i$) can be computed in $O(n)$ time.

The algorithm chooses a random node $i$ and computes gain($i$). If gain($i$) $> 0$, $i$ is incorporated in the current solution and nodes are reassigned as well as facilities removed, if required, so that $F + C$ decreases by gain($i$). This step is performed repeatedly. Note that at any point, demand nodes need not be assigned to the closest facility in the current solution. This may happen because the only reassignments we allow are to the newly added facility. When a new facility is added and existing facilities removed, reassigning to the new facility need not be the optimal thing to do. However, we do not reoptimize at this stage as this could take $O(n^2)$ time. Our analysis goes through for our seemingly suboptimal procedure. The reoptimization, if required, will be performed later if a facility $i$ already in $\mathcal{F}$ is *added* to the solution by the local search procedure. In fact, this is the reason that node $i$ is chosen from amongst all the nodes, instead of nodes not in $\mathcal{F}$.

We will compare the cost of the solution produced by the algorithm with the cost of an arbitrary feasible solution to the facility location LP (see [33, 9, 19]). The set of all feasible solutions to the LP includes all integral solutions to the facility location problem.

LEMMA 2.1. *The cost $F + C$ for the initial solution is at most $n^2 F_{SOL} + nC_{SOL}$, where $F_{SOL}$ and $C_{SOL}$ are the facility cost and service cost of an arbitrary solution $SOL$ to the facility location LP.*

*Proof.* Consider the solution $SOL$. Let $\mathcal{F}$ be the set of facilities $i$ such that $y_i \geq 1/n$. Note that $\mathcal{F}$ must be nonempty. Suppose the most expensive facility in $\mathcal{F}$ has cost $f$. Then $F_{SOL} \geq f/n$. We claim that every demand node $j$ must draw at least $1/n$ fraction of its service from the facilities in $\mathcal{F}$. Let $C_{\mathcal{F}}(j)$ be the minimum distance of demand node $j$ from a facility in $\mathcal{F}$ and let $C_{SOL}(j)$ be the service cost of $j$ in the solution $SOL$. Then $C_{SOL}(j) \geq \frac{1}{n} C_{\mathcal{F}}(j)$. Examine the facilities in increasing order of their facility cost and let $x$ be the last location in this order where a facility of cost $\leq f$ occurs. Then the solution that consists of the first $x$ facilities in this order contains all the facilities in $\mathcal{F}$. Thus, the service cost of this solution is at most $\sum_j C_{\mathcal{F}}(j) \leq n \sum_j C_{SOL}(j) = nC_{SOL}$. Also, the facility cost of this solution is at most $x \cdot f \leq n \cdot f \leq n^2 F_{SOL}$. Hence the cost $C + S$ for this solution is at most $n^2 F_{SOL} + nC_{SOL}$. Since this is one of the solutions considered in choosing an initial solution, the lemma follows. □

LEMMA 2.2. *The initial solution can be chosen in $O(n^2)$ time.*

*Proof.* First, we sort the facilities in increasing order of their facility cost. This takes $O(n \log n)$ time. We compute the costs of candidate solutions in an incremental fashion as follows. We maintain the cost of the solution consisting of the first $i$ facilities in this order (together with assignments of nodes to facilities). From this, we compute the cost of the solution consisting of the first $i+1$ facilities (together with assignments of nodes to facilities). The idea is that the solutions for $i$ and $i+1$ differ very slightly and the change can be computed in $O(n)$ time. Consider the effect of including the $(i+1)$st facility in the solution for $i$. Some nodes may now have to be connected to the new facility instead of their existing assignment. This is the only type of assignment change which will occur. In order to compute the cost of the solution for $i+1$ (and the new assignments), we examine each demand node $j$ and compare its current service cost with the distance of $j$ to the new facility. If it is cheaper to connect $j$ to the new facility, we do so. Clearly, this takes $O(n)$ time.

The initial solution consists of just the cheapest facility. Its cost can be computed in $O(n)$ time. Thus, the cost of the $n$ candidate solutions can be computed in $O(n^2)$ time. The lemma follows. $\square$

LEMMA 2.3. *The function* gain$(i)$ *can be computed in* $O(n)$ *time.*

*Proof.* Let $\mathcal{F}$ be the current set of facilities. For a demand node $j$, let $\sigma(j)$ be the facility in $F$ assigned to $j$. The maximum decrease in $F + C$ resulting from the inclusion of facility $i$ can be computed as follows. Consider each demand node $j$. If the distance of $j$ to $i$ is less than the current service cost of $j$, i.e., $c_{ij} < c_{\sigma(j)j}$, mark $j$ for reassignment to $i$. Let $D$ be the set of demand nodes $j$ such that $c_{ij} < c_{\sigma(j)j}$. The above step marks all the nodes in $D$ for reassignment to the new facility $i$. (Note that none of the marked nodes are actually reassigned; i.e., the function $\sigma$ is not changed in this step. The actual reassignment will occur only if gain$(i)$ is positive.) Having considered all the demand nodes, we consider all the facilities in $F$. Let $i'$ be the currently considered facility. Let $D(i')$ be the set of demand nodes $j$ that are currently assigned to $i'$, i.e., $D(i') = \{j : \sigma(j) = i'\}$. Note that some of the nodes that are currently assigned to $i'$ may have already been marked for reassignment to $i$. Look at the remaining unmarked nodes (possibly none) assigned to $i'$. Consider the change in cost if all these nodes are reassigned to $i$ and facility $i'$ removed from the current solution. The change in the solution cost as a result of this is $-f_{i'} + \sum_{j \in D(i') \setminus D}(c_{ij} - c_{i'j})$. If this results in a decrease in the cost, mark all such nodes for reassignment to $i$ and mark facility $i'$ for removal from the solution. After all the facilities in $F$ have been considered as above, we actually perform all the reassignments and facility deletions, i.e., reassign all marked nodes to $i$ and delete all marked facilities. Then gain$(i)$ is simply $(C_1 + S_1) - (C_2 + S_2)$, where $C_1, S_1$ are the facility and service costs of the initial solution and $C_2, S_2$ are the facility and service costs of the final solution. If this difference is $< 0$, gain$(i)$ is 0.

Now we prove that the above procedure is correct. Suppose that there is some choice of reassignments of demand nodes and facilities in $\mathcal{F}$ to remove such that the gain is more than gain$(i)$ computed above. It is easy to show that this cannot be the case. $\square$

Lemmas 2.6 and 2.7 relate the sum of the gains to the difference between the cost of the current solution and that of an arbitrary fractional solution. For ease of understanding, before proving the results in their full generality, we first prove simpler versions of the lemmas where the comparison is with an arbitrary integral solution.

LEMMA 2.4. $\sum$ gain$(i) \geq C - (F_{SOL} + C_{SOL})$, *where* $F_{SOL}$ *and* $C_{SOL}$ *are the facility and service costs for an arbitrary integral solution.*

*Proof.* Let $\mathcal{F}_{SOL}$ be the set of facilities in solution $SOL$. For a demand node $j$, let $\sigma(j)$ be the facility assigned to $j$ in the current solution and let $\sigma_{SOL}(j)$ be the facility assigned to $j$ in $SOL$. We now proceed with the proof.

With every facility $i \in \mathcal{F}_{SOL}$, we will associate a modified solution as follows. Let $D_{SOL}(i)$ be the set of all demand nodes $j$ which are assigned to $i$ in $SOL$. Consider the solution obtained by including facility $i$ in the current solution and reassigning all nodes in $D_{SOL}(i)$ to $i$. Let $\text{gain}'(i)$ be the decrease in cost of the solution as a result of this modification, i.e., $\text{gain}'(i) = -f_i + \sum_{j \in D_{SOL}(i)} (c_{\sigma(j)j} - c_{ij})$. Note that $\text{gain}'(i)$ could be $< 0$. Clearly, $\text{gain}(i) \geq \text{gain}'(i)$. We will prove that $\sum_{i \in \mathcal{F}_{SOL}} \text{gain}'(i) = C - (F_{SOL} + C_{SOL})$. Notice that for $j \in D_{SOL}(i)$, we have $i = \sigma_{SOL}(j)$. Therefore

$$\sum_{i \in \mathcal{F}_{SOL}} \text{gain}'(i) = \sum_{i \in \mathcal{F}_{SOL}} -f_i + \sum_{i \in \mathcal{F}_{SOL}} \sum_{j \in D_{SOL}(i)} (c_{\sigma(j)j} - c_{\sigma_{SOL}j}).$$

The first term evaluates to $-F_{SOL}$. The summation over the indices $i, j \in D_{SOL}(i)$ can be replaced simply by a sum over the demand points $j$. The summand therefore simplifies to two terms, $\sum_j c_{\sigma(j)j}$, which evaluates to $C$, and $-\sum_j c_{\sigma_{SOL}j}$, which evaluates to $-C_{SOL}$. Putting it all together, we get $\sum_i \text{gain}'(i) = -F_{SOL} + C - C_{SOL}$, which proves the lemma. ∎

LEMMA 2.5. $\sum \text{gain}(i) \geq F - (F_{SOL} + 2C_{SOL})$, *where* $F_{SOL}$ *and* $C_{SOL}$ *are the facility and service costs for an arbitrary integral solution.*

*Proof.* The proof will proceed along lines similar to the proof of Lemma 2.4. As before let $\mathcal{F}$ be the set of open facilities in the current solution. Let $\mathcal{F}_{SOL}$ be the set of facilities in solution $SOL$. For a demand node $j$, let $\sigma(j)$ be the facility assigned to $j$ in the current solution and let $\sigma_{SOL}(j)$ be the facility assigned to $j$ in $SOL$. For a facility $i \in \mathcal{F}$, let $D(i)$ be the set of demand nodes assigned to $i$ in the current solution. For a facility $i \in \mathcal{F}_{SOL}$, let $D_{SOL}(i)$ be the set of all demand nodes $j$ which are assigned to $i$ in $SOL$.

First, we associate every node $i' \in \mathcal{F}$ with its closest node $m(i') \in \mathcal{F}_{SOL}$. For $i \in \mathcal{F}_{SOL}$, let $R(i) = \{i' \in \mathcal{F} | m(i') = i\}$. With every facility $i \in \mathcal{F}_{SOL}$, we will associate a modified solution as follows. Consider the solution obtained by including facility $i$ in the current solution and reassigning all nodes in $D_{SOL}(i)$ to $i$. Further, for all facilities $i' \in R(i)$, the facility $i'$ is removed from the solution and all nodes in $D(i') \setminus D_{SOL}(i)$ are reassigned to $i$. Let $\text{gain}'(i)$ be the decrease in cost of the solution as a result of this modification, i.e.,

$$\text{gain}'(i) = -f_i + \sum_{j \in D_{SOL}(i)} (c_{\sigma(j)j} - c_{ij}) + \sum_{i' \in R(i)} \left( f_{i'} + \sum_{j \in D(i') \setminus D_{SOL}(i)} (c_{\sigma(j)j} - c_{ij}) \right). \tag{2.1}$$

Note that $\text{gain}'(i)$ could be $< 0$. Clearly, $\text{gain}(i) \geq \text{gain}'(i)$. We will prove that $\sum_{i \in \mathcal{F}_{SOL}} \text{gain}'(i) \geq F - (F_{SOL} + 2C_{SOL})$. In order to obtain a bound, we need an upper bound on the distance $c_{ij}$. From the triangle inequality, $c_{ij} \leq c_{i'j} + c_{i'i}$. Since $i' \in R(i)$, $m(i') = i$; i.e., $i$ is the closest node to $i'$ in $\mathcal{F}_{SOL}$. Hence, $c_{i'i} \leq c_{i'\sigma_{SOL}(j)} \leq c_{i'j} + c_{\sigma_{SOL}(j)j}$, where the last inequality follows from triangle inequality. Substituting this bound for $c_{i'i}$ in the inequality for $c_{ij}$, we get $c_{ij} \leq 2c_{i'j} + c_{\sigma_{SOL}(j)j} = 2c_{\sigma(j)j} + c_{\sigma_{SOL}(j)j}$, where the equality comes from the fact that $i' = \sigma(j)$. Substituting this bound for $c_{ij}$ in the last term of (2.1), we get

$$\begin{aligned}\text{gain}'(i) \geq -f_i + &\sum_{j \in D_{SOL}(i)} (c_{\sigma(j)j} - c_{ij}) \\ &+ \sum_{i' \in R(i)} \left( f_{i'} + \sum_{j \in D(i') \setminus D_{SOL}(i)} -(c_{\sigma(j)j} + c_{\sigma_{SOL}(j)j}) \right).\end{aligned}$$

The last term in the sum is a sum over negative terms, so if we sum over a larger set $j \in D(i')$ instead of $j \in D(i') \setminus D_{SOL}(i)$, we will still have a lower bound on $\text{gain}'(i)$:

$$\text{gain}'(i) \geq -f_i + \sum_{j \in D_{SOL}(i)} (c_{\sigma(j)j} - c_{ij}) + \sum_{i' \in R(i)} f_{i'} - \sum_{i' \in R(i)} \sum_{j \in D(i')} (c_{\sigma(j)j} + c_{\sigma_{SOL}(j)j}).$$

The first term in the expression $\sum_i \text{gain}'(i)$ is equal to $-F_{SOL}$, the second is $C - C_{SOL}$ since the double summation over indices $i$ and $j \in D_{SOL}(i)$ is a summation over all the demand nodes $j$, and $\sum_j c_{\sigma(j)j}$ is $C$ and $\sum_j c_{\sigma_{SOL}(j)j}$ is $C_{SOL}$. The third term is the sum of the facility costs of all the nodes in the current solution, which is $F$. The fourth term (which is negative) is equal to $-(C + C_{SOL})$ since the summation over the indices $i, i' \in R(i)$ and $j \in D(i')$ amounts to a summation over all the demand nodes $j$. Therefore we have

$$\sum_{i \in \mathcal{F}_{SOL}} \text{gain}'(i) \geq -F_{SOL} + (C - C_{SOL}) + F - (C + C_{SOL}),$$

which proves the lemma. $\square$

We now claim Lemmas 2.6 and 2.7 to compare with the cost of an arbitrary fractional solution to the facility location LP instead of an integral solution. We present the proofs in section 6 for a smoother presentation.

LEMMA 2.6. *$\sum \text{gain}(i) \geq C - (F_{SOL} + C_{SOL})$, where $F_{SOL}$ and $C_{SOL}$ are the facility and service costs for an arbitrary fractional solution SOL to the facility location LP.*

Similar to the above lemma, we generalize Lemma 2.5 to apply to any fractional solution of the facility location LP. See section 6 for the proof of the lemma.

LEMMA 2.7. *$\sum \text{gain}(i) \geq F - (F_{SOL} + 2C_{SOL})$, where $F_{SOL}$ and $C_{SOL}$ are the facility and service costs for an arbitrary fractional solution SOL to the facility location LP.*

Therefore if we are at a local optimum, where $\text{gain}(i) = 0$ for all $i$, the previous two lemmas guarantee that the facility cost $F$ and service cost $C$ satisfy

$$F \leq F_{SOL} + 2C_{SOL} \qquad \text{and} \qquad C \leq F_{SOL} + C_{SOL}.$$

Recall that a single improvement step of the local search algorithm consists of choosing a random vertex and attempting to improve the current solution; this takes $O(n)$ time. The algorithm will be: *at every step choose a random vertex, compute the possible improvement on adding this vertex (if the vertex is already in the solution, it has cost* 0*), and update the solution if there is a positive improvement.* We can easily argue that the process of computing the $\text{gain}(i)$ for a vertex can be performed in linear time. We now bound the number of improvement steps the algorithm needs to perform until it produces a low cost solution, which will fix the number of iterations we perform in the above steps. We will start by proving the following lemma.

LEMMA 2.8. *After $O(n\log(n/\epsilon))$ iterations we have $C+F \le 2F_{SOL}+3C_{SOL}+\epsilon(F_{SOL}+C_{SOL})$ with probability at least $\frac{1}{2}$.*

*Proof.* Suppose that after $s$ steps $C+F \ge 2F_{SOL}+3C_{SOL}+\epsilon(F_{SOL}+C_{SOL})/e$. Since the local search process decreases the cost monotonically, this implies that in all the intermediate iterations, the above condition on $C+F$ holds.

From Lemmas 2.4 and 2.5, we have

$$\sum_i \text{gain}(i) \ge \frac{1}{2}(C+F-(2F_{SOL}+3C_{SOL})).$$

Let $g(i)=\text{gain}(i)/(C+F-(2F_{SOL}+3C_{SOL}))$. Then in all intermediate iterations $\sum_i g(i) \ge \frac{1}{2}$. Let $P_t$ be the value of $C+F-(2F_{SOL}+3C_{SOL})$ after $t$ steps. Let $\text{gain}_t(i)$ and $g_t(i)$ be the values of $\text{gain}(i)$ and $g(i)$ at the $t$th step. Observe that $E[g_t(i)] \ge \frac{1}{2n}$.

Suppose $i$ is the node chosen for step $t+1$. Then, *assuming* $P_t > 0$,

$$\begin{aligned} P_{t+1} &= P_t - \text{gain}_t(i) = P_t(1-g_t(i)), \\ \frac{eP_{t+1}}{\epsilon(F_{SOL}+C_{SOL})} &= \frac{eP_t}{\epsilon(F_{SOL}+C_{SOL})}(1-g_t(i)), \\ \ln\left(\frac{eP_{t+1}}{\epsilon(F_{SOL}+C_{SOL})}\right) &= \ln\left(\frac{eP_t}{\epsilon(F_{SOL}+C_{SOL})}\right) + \ln(1-g_t(i)) \\ &\le \ln\left(\frac{eP_t}{\epsilon(F_{SOL}+C_{SOL})}\right) - g_t(i). \end{aligned}$$

Let $Q_t = \ln((eP_t)/(\epsilon(F_{SOL}+C_{SOL})))$. Then $Q_{t+1} \le Q_t - g_t(i)$. Note that the node $i$ is chosen uniformly and at random from amongst $n$ nodes. Our initial assumption implies that $P_t > \epsilon(F_{SOL}+C_{SOL})/e$ for all $t \le s$; hence, $Q_t > 0$ for all $t \le s$. Applying linearity of expectation,

$$E[Q_{s+1}] \le E[Q_s] - \frac{1}{2n} \le \cdots \le Q_1 - \frac{s}{2n}.$$

Note that the initial value $Q_1 \le \ln(en/\epsilon)$. So if $s = 2n\ln\frac{n}{\epsilon} - n$, then $E[Q_{s+1}] \le \frac{1}{2}$. By the Markov inequality, $\text{Prob}[Q_{s+1} \ge 1] \le \frac{1}{2}$. If $Q_{s+1} \le 1$, then $P_{s+1} \le \epsilon(F_{SOL}+C_{SOL})$, which proves the lemma. ▯

THEOREM 2.9. *The algorithm produces a solution such that $F \le (1+\epsilon)(F_{SOL}+2C_{SOL})$ and $C \le (1+\epsilon)(F_{SOL}+C_{SOL})$ in $O(n(\log n+\frac{1}{\epsilon}))$ steps (running time $O(n^2(\log n+\frac{1}{\epsilon}))$) with constant probability.*

*Proof.* We are guaranteed by Lemma 2.8 that after $O(n\log(n/\epsilon))$ steps we have the following:

$$C+F \le 2F_{SOL}+3C_{SOL}+\epsilon(F_{SOL}+C_{SOL}).$$

Once the above condition is satisfied, we say that we begin the second phase. We have already shown that with probability at least 1/2, we require $O(n\log(n/\epsilon))$ steps for the first phase.

We stop the analysis as soon as both $C \le F_{SOL}+C_{SOL}+\epsilon(F_{SOL}+C_{SOL})$ and $F \le F_{SOL}+2C_{SOL}+\epsilon(F_{SOL}+C_{SOL})$. At this point we declare that the second phase has ended.

Suppose one of these is violated. Then applying either Lemma 2.4 or Lemma 2.5, we get $\sum_i \text{gain}(i) \ge \epsilon(F_{SOL}+C_{SOL})$.

Define $\text{gain}_t(i)$ to be value of $\text{gain}(i)$ for a particular step $t$. Assuming the conditions of the theorem are not true for a particular step $t$ in the second phase, we have $E[\text{gain}_t(i)] \geq \frac{\epsilon}{n}(F_{SOL} + C_{SOL})$. Let the assignment and facility costs be $C_t$ and $F_t$ after $t$ steps into the second phase. Assuming we did not satisfy the conditions of the theorem in the $t$th step,

$$C_{t+1} + F_{t+1} \leq C_t + F_t - \text{gain}_t(i).$$

At the beginning of the second phase, we have

$$\begin{aligned} F_1 + C_1 &\leq 2F_{SOL} + 3C_{SOL} + \epsilon(F_{SOL} + C_{SOL}) \\ &\leq 3(F_{SOL} + C_{SOL}) + \epsilon(F_{SOL} + C_{SOL}). \end{aligned}$$

Conditioned on the fact that the second phase lasts for $s$ steps,

$$E[C_{s+1} + F_{s+1}] \leq 3(F_{SOL} + C_{SOL}) - \frac{s\epsilon}{n}(F_{SOL} + C_{SOL}).$$

Setting $s = n + 5n/(2\epsilon)$, we get $E[C_{s+1} + F_{s+1}] \leq (F_{SOL} + C_{SOL})/2$.

Observe that if $F_{SOL}, C_{SOL}$ are the facility and service costs for the optimum solution, we have obtained a contradiction to the fact that we can make $s$ steps without satisfying the two conditions. However, since we are claiming the theorem for any feasible solution $SOL$, we have to use a different argument.

If $E[C_{s+1} + F_{s+1}] \leq (F_{SOL} + C_{SOL})/2$, and since $C + F$ is always positive, with probability at least $\frac{1}{2}$ we have $C_{s+1} + F_{s+1} \leq (F_{SOL} + C_{SOL})$. In this case both conditions of the theorem are trivially true.

Thus with probability at least 1/4 we take $O(n \log(n/\epsilon) + n/\epsilon)$ steps. Assuming $1/\epsilon > \ln(1/\epsilon)$, we can drop the $\ln \epsilon$ term and claim the theorem. ∎

Following standard techniques, the above theorem yields a high probability result, losing another factor of $\log n$ in running time. The algorithm can be derandomized very easily, at the cost of a factor $n$ increase in the running time. Instead of picking a random node, we try all the $n$ nodes and choose the node that gives the maximum gain. Each step now takes $O(n^2)$ time. The number of steps required by the deterministic algorithm is the same as the expected number of steps required by the randomized algorithm.

THEOREM 2.10. *The deterministic algorithm produces a solution such that* $F \leq (1+\epsilon)(F_{SOL} + 2C_{SOL})$ *and* $C \leq (1+\epsilon)(F_{SOL} + C_{SOL})$ *in* $O(n(\log n + \frac{1}{\epsilon}))$ *steps with running time* $O(n^3(\log n + \frac{1}{\epsilon}))$.

## 3. Scaling costs.

In this section we will show how cost scaling can be used to show a better approximation guarantee. The idea of scaling exploits the asymmetry in the guarantees of the service cost and facility cost.

We will scale the facility costs uniformly by some factor (say $\delta$). We will then solve the modified instance using local search (or in later sections by a suitable algorithm). The solution of the modified instance will then be scaled back to determine the cost in the original instance.

*Remark.* Notice that the way the proof of Theorem 2.9 is presented, the termination condition of the algorithm that runs on the scaled instance is not obvious. We will actually run the algorithm for $O(n(\log n + 1/\epsilon))$ steps and construct that many different solutions. The analysis shows that with constant probability we find a solution such that both of the conditions are satisfied for at least one of these $O(n(\log n + 1/\epsilon))$ solutions. We will scale back all of these solutions and use the best solution. The same results will carry over with a high probability guarantee at the cost of another $O(\log n)$ in the running time. We first claim the following simple theorem.

THEOREM 3.1. *The uncapacitated facility location problem can be approximated to factor $1+\sqrt{2}+\epsilon$ in randomized $O(n^2/\epsilon+n^2\log n)$ time.*

*Proof.* Assume that the facility cost and the service costs of the optimal solution are denoted by $F_{OPT}$ and $C_{OPT}$. Then after scaling, there exists a solution to the modified instance of modified facility cost $\delta F_{OPT}$ and service cost $C_{OPT}$. For some small $\epsilon'$ we have a solution to the scaled instance, with service cost $C$ and facility cost $F$ such that

$$F \le (1+\epsilon')(\delta F_{OPT}+2C_{OPT}), \qquad C \le (1+\epsilon')(\delta F_{OPT}+C_{OPT}).$$

Scaling back, we will have a solution of the same service cost and facility cost $F/\delta$. Thus the total cost of this solution will be

$$C+F/\delta \le (1+\epsilon')\left[(1+\delta)F_{OPT}+\left(1+\frac{2}{\delta}\right)C_{OPT}\right].$$

Clearly setting $\delta=\sqrt{2}$ gives a $1+\sqrt{2}+\epsilon$ approximation, where $\epsilon=(1+\sqrt{2})\epsilon'$. ☐

Actually the above algorithm used only the fact that there existed a solution of a certain cost. In fact, the above proof will go through for any solution, even fractional. Let the facility cost of such a solution be $F_{SOL}$ and its service cost $C_{SOL}$. The guarantee provided by the local search procedure after scaling back yields facility cost $\hat{F}$ and service cost $\hat{C}$ such that

$$\hat{F} \le (1+\epsilon)(F_{SOL}+2C_{SOL}/\delta), \qquad \hat{C} \le (1+\epsilon)(\delta F_{SOL}+C_{SOL}).$$

Setting $\delta=2C_{SOL}/(\gamma F_{SOL})$, we get that the facility cost is at most $(1+\gamma)F_{SOL}$ and the service cost is $(1+2/\gamma)C_{SOL}$ up to factors of $(1+\epsilon)$ for arbitrarily small $\epsilon$.

In fact, the costs $F_{SOL}, C_{SOL}$ can be guessed up to factors of $1+\epsilon'$, and we will run the algorithm for all the resulting values of $\delta$. We are guaranteed to run the algorithm for some value of $\delta$ which is within a $1+\epsilon'$ factor of $2C_{SOL}/(\gamma F_{SOL})$. For this setting of $\delta$ we will get the result claimed in the theorem. This factor of $(1+\epsilon')$ can be absorbed by the $1+\epsilon$ term associated with the tradeoff.

Notice that we can guess $\delta$ directly and run the algorithm for all guesses. Each guess would return $O(n(\log n+\frac{1}{\epsilon}))$ solutions, such that one of them satisfies both the bounds (for that $\delta$). Thus if we consider the set of all solutions over all guesses of $\delta$, one of the solutions satisfies the theorem—in some sense we can achieve the tradeoff in an oblivious fashion. Of course this increases the running time of the algorithm appropriately. We need to ensure that, for every guess $\delta$, one of the $O(n(\log n+\frac{1}{\epsilon}))$ solutions satisfies the bounds on the facility cost and service cost.

THEOREM 3.2. *Let SOL be any solution to the facility location problem (possibly fractional) with facility cost $F_{SOL}$ and service cost $C_{SOL}$. For any $\gamma>0$, the local search heuristic proposed (together with scaling) gives a solution with facility cost at most $(1+\gamma)F_{SOL}$ and service cost at most $(1+2/\gamma)C_{SOL}$. The approximation is up to multiplicative factors of $(1+\epsilon)$ for arbitrarily small $\epsilon>0$.*

The known results on the tradeoff problem use a $(p,q)$ notation where the first parameter $p$ denotes the approximation factor of the facility cost and $q$ denotes the approximation factor for the service cost. This yields a better tradeoff for the $k$-median problem than the tradeoff $(1+\gamma, 2+2/\gamma)$ given by Lin and Vitter [25] as well as the tradeoff $(1+\gamma, 3+5/\gamma)$ given by Korupolu, Plaxton, and Rajaraman [23]. For facility location, our tradeoff is better than the tradeoff of $(1+\gamma, 3+3/\gamma)$ obtained by Shmoys, Tardos, and Aardal [33]. We note that the techniques of Marathe et al. [29] for bicriteria approximations also yield tradeoffs for facility cost versus service cost. Their results are stated in terms of tradeoffs for similar objectives, e.g., (cost, cost) or (diameter, diameter) under two different cost functions. However, their parametric search algorithm will also yield tradeoffs for different objective functions provided there exists a $\rho$ approximation algorithm to minimize the sum of the two objectives. By scaling the cost functions, their algorithm produces a tradeoff of $(\rho(1+\gamma), \rho(1+1/\gamma))$. For tradeoffs of facility cost versus service cost, $\rho$ is just the approximation ratio for facility location.

The above tradeoff is also interesting since the tradeoff of $(1+\gamma, 1+1/\gamma)$ is the best tradeoff possible; i.e., we cannot obtain a $(1+\gamma-\epsilon, 1+1/\gamma-\delta)$ tradeoff for any $\epsilon, \delta > 0$. This is illustrated by a very simple example.

**3.1. Lower bound for tradeoff.** We present an example to prove that the $(1+\gamma, 1+2/\gamma)$ tradeoff between facility and service costs is almost the best possible when comparing with a fractional solution of the facility location LP. Consider the following instance. The instance $\mathcal{I}$ consists of two nodes $u$ and $v$, $c_{uv} = 1$. The facility costs are given by $f_u = 1$, $f_v = 0$. The demands of the nodes are $d_u = 1$, $d_v = 0$.

THEOREM 3.3. *For any $\gamma > 0$, there exists a fractional solution to $\mathcal{I}$ with facility cost $F_{SOL}$ and service cost $C_{SOL}$ such that there is no integral solution with facility cost strictly less than $(1+\gamma)F_{SOL}$ and service cost strictly less than $(1+1/\gamma)C_{SOL}$.*

*Proof.* Observe that there are essentially two integral solutions to $\mathcal{I}$. The first, $SOL_1$, chooses $u$ as a facility, $F_{SOL_1} = 1$, $C_{SOL_1} = 0$. The second, $SOL_2$, chooses $v$ as a facility, $F_{SOL_2} = 0$, $C_{SOL_2} = 1$. For $\gamma > 0$, we will construct a fractional solution for $\mathcal{I}$ such that $F_{SOL} = 1/(1+\gamma)$, $C_{SOL} = \gamma/(1+\gamma)$. The fractional solution is obtained by simply taking the linear combination $(1/(1+\gamma))SOL_1 + (\gamma/(1+\gamma))SOL_2$. It is easy to verify that this satisfies the conditions of the lemma. ∎

Theorem 3.3 proves that a tradeoff of $(1+\gamma, 1+1/\gamma)$ for facility cost versus service cost is the best possible.

**3.2. Scaling and capacitated facility location.** The scaling idea can also be used to improve the approximation ratio for the capacitated facility location problem. Chudak and Williamson [12] prove that a local search algorithm produces a solution with the following properties (modified slightly from their exposition):

$$\begin{aligned} C &\leq (1+\epsilon')(F_{OPT} + C_{OPT}), \\ F &\leq (1+\epsilon')(5F_{OPT} + 4C_{OPT}). \end{aligned}$$

This gives a $6+\epsilon$ approximation for the problem. However, we can exploit the asymmetric guarantee by scaling. Scaling the facility costs by a factor $\delta$, we get a solution for the scaled instance such that

$$\begin{aligned} C &\leq (1+\epsilon')(\delta F_{OPT} + C_{OPT}), \\ F &\leq (1+\epsilon')(5\delta F_{OPT} + 4C_{OPT}). \end{aligned}$$

Scaling back, we get a solution of cost $C + F/\delta$. But

$$C + F/\delta \leq (1+\epsilon')\left[(5+\delta)F_{OPT} + \left(1+\frac{4}{\delta}\right)C_{OPT}\right].$$

Setting $\delta = 2\sqrt{2}-2$, we get a $3+2\sqrt{2}+\epsilon$ approximation.

**4. Primal-dual algorithm and improvements.** In this section we will show that the ideas of a primal-dual algorithm can be combined with augmentation and scaling to give a result better than those obtained by the pure greedy strategy and the primal-dual algorithm itself.

We will not be presenting the details of the primal-dual algorithm here, since we would be using the algorithm as a "black box." See [19, 7] for the primal-dual algorithm and its improvements. The following lemma is proved in [19].

LEMMA 4.1 (see [19]). *The primal-dual algorithm returns a solution with facility cost $F$ and a service cost $S$ such that $3F + S \leq 3OPT$, where $OPT$ denotes the cost of the optimal dual solution. The algorithm runs in time $O(n^2 \log n)$.*

In itself the primal-dual algorithm does not yield a better result than factor 3. In fact, a simple example shows that the dual constructed, which is used as a lower bound to the optimum, can be factor 3 away from the optimum. We will further introduce the notation that the facility cost of the optimal solution be $F_{OPT}$ and the service cost be $C_{OPT}$. *We would like to observe that these quantities are only used in the analysis.*

Let us first consider a simpler algorithm before presenting the best known combinatorial algorithm. If in the primal-dual algorithm we were to scale facility costs by a factor of $\delta = 1/3$ and use the primal-dual algorithm on this modified instance, we would have a feasible primal solution of cost $F_{OPT}/3 + C_{OPT}$. The primal-dual algorithm giving a solution with modified facility cost $F$ and service cost $C$ will guarantee that $3F + S$ is at most 3 times the feasible dual constructed which is less than the feasible primal solution of $F_{OPT}/3 + C_{OPT}$. After scaling back the solution, the cost of the final solution will be $3F + S$ due to the choice of $\delta$, which is at most $F_{OPT} + 3C_{OPT}$.

Now along with this compare the local search algorithm with $\delta = 2$ for which the cost of the final solution is $3F_{OPT} + 2C_{OPT}$ from the proof of Theorem 3.1. The smaller of the two solutions can be at most $2\frac{1}{3}$ times the optimum cost, which is $F_{OPT} + C_{OPT}$.

COROLLARY 4.2. *Using augmentation and scaling along with the primal-dual algorithm, the facility location problem can be approximated within a factor of $2\frac{1}{3} + \epsilon$ in time $O(n^2/\epsilon + n^2 \log n)$.*

**4.1. Gap examples for dual solutions of the primal-dual algorithm.** We present an example to show that the primal-dual algorithm for facility location can construct a dual whose value is $3 - \epsilon$ away from the optimal for arbitrarily small $\epsilon$. This means that it is not possible to prove an approximation ratio better than $3 - \epsilon$ using the dual constructed as a lower bound. This lower bound for the primal-dual algorithm is the analog of an integrality gap for LPs.

The example is shown in Figure 4.1. Here $r$ is a parameter. The instance is defined on a tree rooted at $w'$. $w'$ has a single child $w$ at unit distance from it. $w$ has $r^2$ children $v_1, \ldots, v_{r^2}$, each at unit distance from $w$. Further, each $v_i$ has a single child $v_i'$ at unit distance from it. All other distances are shortest path distances along the tree. Node $w'$ has a facility cost of 2, and the nodes $v_i$ have facility costs of $2 + 2/r$; all other nodes have facility cost of $\infty$. The node $w$ has demand $2r$. Each $v_i'$ has unit demand, and the rest of the nodes have zero demand. In the dual solution constructed by the algorithm, the $\alpha$ value of $w$ is $2r(1 + 1/r)$ and the $\alpha$ value for each $v_i'$ is $1 + 2/r$. The value of the dual solution is $r^2 + 4r + 2$. However, the value of the optimal solution is $3r^2 + 2r + 2/r$ (corresponding to choosing one of the facilities $v_i$). The ratio of the optimal solution value to dual solution value exceeds $3 - \epsilon$ for $r$ suitably large.

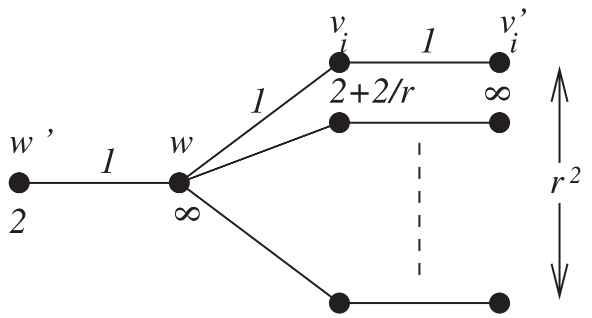


FIG. 4.1. *Lower bound example for the primal-dual facility location algorithm.*

**4.2. Greedy strategy.** We will use a slightly different augmentation technique as described in [16] and improve the approximation ratio. We will use a lemma proved in [16] about greedy augmentation; however the lemma proved therein was not cast in a form which we can use. The lemma as stated was required to know the value of the optimal facility cost. However, the lemma can be proved for any feasible solution by only assuming its existence. This also shows that the modification to the LP in [16] is not needed. We provide a slightly simpler proof.

The greedy augmentation process proceeds iteratively. At iteration $i$ we pick a node $u$ of cost $f_u$ such that if the service cost decreases from $C_i$ to $C'$ after opening $u$, the ratio $\frac{C_i - C' - f_u}{f_u}$ is maximized. Notice that this is gain$(u)/f_u$, where gain$(u)$ is defined as in section 2. That section used the node with the largest gain to get a fast algorithm; here we will use the best ratio gain to get a better approximation factor.

Assume we start with a solution of facility cost $F$ and service cost $C$. Initially $F_0 = F$ and $C_0 = C$. The node which has the maximum ratio can be found in time $O(n^2)$, and since no node will be added twice, the process will take time at most $O(n^3)$. We assume that the cost of all facilities is nonzero, since facilities with zero cost can always be included in any solution as a preprocessing step.

LEMMA 4.3 (see [16]). *If SOL is a feasible (fractional) solution and the initial solution has facility cost $F$ and service cost $C$, then after greedy augmentation the solution cost is at most*

$$F + F_{SOL} \max\left[0, \ln\left(\frac{C - C_{SOL}}{F_{SOL}}\right)\right] + F_{SOL} + C_{SOL}.$$

*Proof.* If the initial service cost is less than or equal to $F_{SOL} + C_{SOL}$, the lemma is true by observation.

Assume that at a point the current service cost $C_i$ is more than $F_{SOL} + C_{SOL}$. The proof of Lemma 2.6 shows that if $\sum_i y_i \text{gain}(i) \geq C_i - C_{SOL} - F_{SOL}$ and $\sum_i y_i f_i = F_{SOL}$, we are guaranteed to have a node with ratio at least $\frac{C_i - C_{SOL} - F_{SOL}}{F_{SOL}}$. Let the facility cost be denoted by $F_i$ at iteration $i$. We are guaranteed

$$\frac{C_i - C_{i+1} - (F_{i+1} - F_i)}{F_{i+1} - F_i} \geq \frac{C_i - C_{SOL} - F_{SOL}}{F_{SOL}}.$$

This equation rearranges to (assuming $C_i > C_{SOL} + F_{SOL}$)

$$F_{i+1} - F_i \leq F_{SOL}\left(\frac{C_i - C_{i+1}}{C_i - C_{SOL}}\right).$$

Suppose that at the $m$th iteration $C_m$ was less than or equal to $F_{SOL}+C_{SOL}$ for the first time. After this point the total cost only decreased. We will upper bound the cost at this step, and the result will hold for the final solution. The cost at this point is

$$F_m + C_m \le F + \sum_{i=1}^{m}(F_i - F_{i-1}) + C_m \le F + F_{SOL}\left(\sum_{i=1}^{m}\frac{C_{i-1}-C_i}{C_{i-1}-C_{SOL}}\right) + C_m.$$

The above expression is maximized when $C_m = F_{SOL}+C_{SOL}$. The derivative with respect to $C_m$ (which is $1-\frac{F_{SOL}}{C_{m-1}-C_{SOL}}$) is positive since $C_{m-1} > C_{SOL}+F_{SOL}$. Thus since $C_m \le F_{SOL}+C_{SOL}$, the boundary point of $C_m = F_{SOL}+C_{SOL}$ gives the maxima. In the following discussion we will assume that $C_m = C_{SOL}+F_{SOL}$.

We use the fact that for all $0 < x \le 1$, we have $\ln(1/x) \ge 1-x$. The cost is therefore

$$\begin{aligned} F + F_{SOL}\sum_{i=1}^{m}\frac{C_{i-1}-C_i}{C_{i-1}-C_{SOL}} + C_m &= F + F_{SOL}\sum_{i=1}^{m}\left(1-\frac{C_i-C_{SOL}}{C_{i-1}-C_{SOL}}\right) + C_m \\ &\le F + F_{SOL}\sum_{i=1}^{m}\ln\left(\frac{C_{i-1}-C_{SOL}}{C_i-C_{SOL}}\right) + C_m. \end{aligned}$$

The above expression for $C_0 = C$ and $C_m = F_{SOL}+C_{SOL}$ proves the lemma. □

**4.3. Better approximations for the facility location problem.** We now describe the full algorithm. Given a facility location problem instance we first scale the facility costs by a factor $\delta$ such that $\ln(3\delta) = 2/(3\delta)$ as in section 3. We run the primal-dual algorithm on the scaled instance. Subsequently, we scale back the solution and apply the greedy augmentation procedure given in [16] and described above. We claim the following.

THEOREM 4.4. *The facility location problem can be approximated within factor* $\approx 1.8526$ *in time* $O(n^3)$.

*Proof.* There exists a solution of cost $\delta F_{OPT}+C_{OPT}$ to the modified problem. Applying the primal-dual algorithm to this gives us a solution of (modified) facility cost $F'$ and service cost $C'$. If we consider this as a solution to the original problem, then the facility cost is $F = F'/\delta$ and the service cost $C = C'$. Now from the analysis of the primal-dual method we are guaranteed

$$3\delta F + C = 3F' + C' \le 3(\delta F_{OPT} + C_{OPT}).$$

If at this point we have $C \le F_{OPT}+C_{OPT}$, then

$$F + C \le \frac{3\delta F + C}{3\delta} + \left(1-\frac{1}{3\delta}\right)C \le \left(2-\frac{1}{3\delta}\right)F_{OPT} + \left(1+\frac{2}{3\delta}\right)C_{OPT}.$$

In the case when $C > F_{OPT}+C_{OPT}$, we also have $C \le 3\delta F_{OPT} - 3\delta F + 3C_{OPT}$. Since there is a solution of service cost $C_{OPT}$ and facility cost $F_{OPT}$, by Lemma 4.3, the cost after greedy augmentation is at most

$$F + F_{OPT}\ \ln\left(\frac{3\delta F_{OPT} - 3\delta F + 2C_{OPT}}{F_{OPT}}\right) + F_{OPT} + C_{OPT}.$$

Over the allowed interval of $F$ the above expression is maximized at $F = (2C_{OPT})/(3\delta)$, with cost

$$(1 + \ln(3\delta))F_{OPT} + \left(1 + \frac{2}{3\delta}\right) C_{OPT}.$$

Now for $\delta > 1/3$ the term $1 + \ln(3\delta)$ is larger than $2 - 1/(3\delta)$. Thus the case $C > F_{OPT} + C_{OPT}$ dominates. Consider the value of $\delta$ such that $\ln(3\delta) = 2/(3\delta)$, which is $\delta \approx 0.7192$, and the approximation factor is $\approx 1.8526$. The greedy augmentation takes maximum $O(n^3)$ time and the primal-dual algorithm takes $O(n^2)$ time. Thus, the theorem follows. ◻

It is interesting, however, that this result which is obtained from the above greedy algorithm can be combined with the algorithm of [9, 10] to obtain a very marginal improvement in the approximation ratio for the facility location problem. The results of [9, 10] actually provide a guarantee that if $\rho$ denotes the fraction of the facility cost in optimal solution returned by the LP formulation of the facility location problem, then the fractional solution can be rounded within a factor of $1+\rho \ln \frac{2}{\rho}$ when $\rho \leq 2/e$. The above primal-dual plus greedy algorithm when run with $\delta = 2/(3(e-1))$ gives an algorithm with approximation ratio $e - \rho(e - 1 - \log \frac{2}{e-1})$. It is easy to verify that the smaller of the two solutions has an approximation of 1.728. This is a very marginal ($\approx 0.008$) improvement over the LP-rounding algorithm. However, it demonstrates that the facility location problem can be approximated better.

THEOREM 4.5. *Combining the LP approach and the scaled, primal-dual plus greedy algorithms, the facility location problem can be approximated to a factor of* 1.728.

## 5. Capacitated facility location.

We consider the following capacitated variant of facility location: A facility at location $i$ is associated with a capacity $u_i$. Multiple facilities can be built at location $i$; each can serve a demand of at most $u_i$. This was considered by Chudak and Shmoys [11] who gave a 3 approximation for the case when all capacities are the same. Jain and Vazirani [19] extended their primal-dual algorithm for facility location to obtain a 4 approximation for the capacitated problem for general capacities. We use the ideas in [19] to improve the approximation ratio for this capacitated variant.

Given an instance $I$ of capacitated facility location, we construct an instance $I'$ of uncapacitated facility location with a modified distance function

$$c'_{ij} = c_{ij} + \frac{f_i}{u_i}.$$

The facility costs in the new instance are the same as the facility costs $f_i$ in the capacitated instance.[2] This construction is similar to the construction in [19].

The folowing lemma relates the assignment and facility costs for the original capacitated instance $I$ to those for the modified uncapacitated instance $I'$.

LEMMA 5.1. *Given a solution $SOL$ to $I$ of assignment cost $C_{SOL}$ and facility cost $F_{SOL}$, there exists a solution of $I'$ of assignment cost at most $C_{SOL} + F_{SOL}$ and facility cost at most $F_{SOL}$.*

*Proof.* Let $y_i$ be the number of facilities built at $i$ in $SOL$. Let $x_{ij}$ be an indicator variable that is 1 if and only if $j$ is assigned to $i$ in $SOL$. $C_{SOL} = \sum_{ij} x_{ij} c_{ij}$ and $F_{SOL} = \sum_i y_i f_i$. The capacity constraint implies that $\sum_j x_{ij} \leq u_i \cdot y_i$. Using $SOL$,

[2]As pointed out by a reviewer, setting $c'_{ji} = c'_{ij}$ preserves metric properties.

we construct a feasible solution $SOL'$ of $I'$ as follows: The assignments of nodes to facilities is the same as in $SOL$. A facility is built at $i$ if and only if at least one facility is built at $i$ in $SOL$.

Clearly the facility cost of $SOL'$ is at most $F_{SOL}$. The assignment cost of $SOL'$ is

$$\begin{aligned}\sum_{ij} x_{ij}c'_{ij} &= \sum_{ij} x_{ij}\left(c_{ij}+\frac{f_i}{u_i}\right)\\ &= \sum_{ij} x_{ij}c_{ij} + \sum_i f_i \frac{\sum_j x_{ij}}{u_i}\\ &\le \sum_{ij} x_{ij}c_{ij} + \sum_i f_i y_i\\ &= C_{SOL} + F_{SOL}. \qquad \square\end{aligned}$$

The following lemma states that given a feasible solution to $I'$, we can obtain a solution to $I$ of no greater cost.

LEMMA 5.2. *Given a feasible solution $SOL'$ to $I'$, there exists a feasible solution $SOL$ to $I$ such that the cost of $SOL$ is at most the cost of $SOL'$.*

*Proof.* Let $Y_i$ be an indicator variable that is 1 if and only if a facility is built at $i$ in $SOL'$. Let $x_{ij}$ be an indicator variable that is 1 if and only if $j$ is assigned to $i$ in $SOL'$. The solution $SOL$ for instance $I$ is obtained as follows: The assignment of nodes to facilities is exactly the same as in $SOL'$. The number of facilities built at $i$ is

$$y_i = \left\lceil \frac{\sum_j x_{ij}}{u_i} \right\rceil .$$

Thus $y_i \le \frac{\sum_j x_{ij}}{u_i} + Y_i$. The cost of $SOL'$ is

$$\begin{aligned}\sum_{ij} x_{ij}c'_{ij} + \sum Y_i f_i &= \sum_{ij} x_{ij}\left(c_{ij}+\frac{f_i}{u_i}\right) + \sum_i Y_i f_i\\ &= \sum_{ij} x_{ij}c_{ij} + \sum_i f_i\left(\frac{\sum_j x_{ij}}{u_i} + Y_i\right)\\ &\ge \sum_{ij} x_{ij}c_{ij} + \sum_i f_i y_i.\end{aligned}$$

Note that the last expression denotes the cost of $SOL$. Thus the cost of $SOL$ is at most the cost of $SOL'$. $\square$

Suppose we have instance $I$ of capacitated facility location. We first construct the modified instance $I'$ of uncapacitated facility location as described above. We will use an algorithm for uncapacitated facility location to obtain a solution to $I'$ and then translate it to a solution to $I$. Let $OPT(I)$ (resp., $OPT(I')$) denote the cost of the optimal solution for instance $I$ (resp., $I'$). First, we show that a $\rho$ approximation for uncapacitated facility location gives a $2\rho$ approximation for the capacitated version.

LEMMA 5.3. *Given a $\rho$ approximation for uncapacitated facility location, we can get a $2\rho$ approximation for the capacitated version.*

*Proof.* Lemma 5.1 implies that $OPT(I') \le 2OPT(I)$. Since we have a $\rho$ approximation algorithm for uncapacitated facility location, we can obtain a solution to $I'$

of cost at most $\rho OPT(I') \leq 2\rho OPT(I)$. Now, Lemma 5.2 guarantees that we can obtain a solution to $I$ of cost at most $2\rho OPT(I)$, yielding a $2\rho$ approximation for the capacitated version. $\square$

The above lemma implies an approximation ratio of $\approx 3.705$ in $O(n^3)$ time using the combinatorial algorithm of section 4.3 and an approximation ratio of $\approx 3.456$ in polynomial time using the guarantee of Theorem 4.5, combining the LP rounding and combinatorial algorithms.

By performing a more careful analysis, we can obtain a slightly better approximation guarantee for the combinatorial algorithm. Let $F_{OPT}$ and $C_{OPT}$ denote the facility and assignment costs for the optimal solution to $I$. Let $F'$ and $C'$ be the facility and assignment costs of the solution to $I'$ guaranteed by Lemma 5.1. Then $F' \leq F_{OPT}$ and $C' \leq F_{OPT} + C_{OPT}$.

LEMMA 5.4. *The capacitated facility location problem can be approximated within factor* $3 + \ln 2 \approx 3.693$ *in time* $O(n^3)$.

*Proof.* We use the approximation algorithm described in section 4.3 and analyzed in Theorem 4.4. We have a feasible solution to $I'$ of facility cost $F'$ and assignment cost $C'$. The proof of Theorem 4.4 bounds the cost of the solution obtained to $I'$ by

$$(1 + \ln(3\delta))F' + \left(1 + \frac{2}{3\delta}\right) C'.$$

Here $\delta$ is a parameter for the algorithm. Substituting the bounds for $F'$ and $C'$ and rearranging, we get the bound

$$\left(2 + \ln(3\delta) + \frac{2}{3\delta}\right) F_{OPT} + \left(1 + \frac{2}{3\delta}\right) C_{OPT}.$$

Setting $\delta = \frac{2}{3}$, so as to minimize the coefficient of $F_{OPT}$, we obtain the bound $(3 + \ln 2)F_{OPT} + 2C_{OPT}$. This yields the claimed approximation ratio. Also, the algorithm runs in $O(n^3)$ time. $\square$

## 6. Proofs of lemmas in section 2.

We present the proofs of the lemmas we omitted to avoid discontinuity in the presentation. Recall that the facility location LP is as follows:

$$\begin{aligned} \min \sum_i y_i f_i + \sum_{ij} x_{ij} c_{ij} & \\ \forall ij \quad x_{ij} &\leq y_i, \\ \forall j \quad \sum_i x_{ij} &\geq 1, \\ x_{ij}, y_i &\geq 0. \end{aligned}$$

Here $y_i$ indicates whether facility $i$ is chosen in the solution, and $x_{ij}$ indicates whether node $j$ is serviced by facility $i$.

LEMMA 2.6. $\sum \text{gain}(i) \geq C - (F_{SOL} + C_{SOL})$, *where* $F_{SOL}$ *and* $C_{SOL}$ *are the facility and service costs for an arbitrary fractional solution SOL to the facility location LP.*

*Proof.* Consider the fractional solution $SOL$ to the facility location LP. $F_{SOL} = \sum_i y_i f_i$ and $C_{SOL} = \sum_{ij} x_{ij} c_{ij}$. We will prove that $\sum_i y_i \cdot \text{gain}(i) \geq C - (F_{SOL} + C_{SOL})$. Assume without loss of generality that $y_i \leq 1$ for all $i$ and $\sum_i x_{ij} = 1$ for all $j$.

We first modify the solution (and make multiple copies of facilities) so as to guarantee that for all $ij$, either $x_{ij} = 0$ or $x_{ij} = y_i$. The pair $ij$ for which this condition is violated is said to be a violating $ij$. The modification is done as follows: Suppose there is a violating $ij$, i.e., there are a facility $i$ and demand node $j$ in our current solution such that $0 < x_{ij} < y_j$. Let $x = \min_j\{x_{ij} | x_{ij} > 0\}$. We create a copy $i'$ of node $i$ (i.e., $c_{i'i} = 0$, $c_{i'j} = c_{ij}$ for all $j$ and $f_{i'} = f_i$.) We set $y_{i'} = x$ and decrease $y_{i'}$ by $x$. Further, for all $j$ such that $x_{ij} > 0$, we decrease $x_{ij}$ by $x$ and set $x_{i'j} = x$. For all the remaining $j$, we set $x_{i'j} = 0$. Note that the new solution we obtain is a fractional solution with the same cost as the original solution. Also, all $i'j$ satisfy the desired conditions. Further, the number of violating $ij$ decreases. In at most $n^2$ such operations, we obtain a solution that satisfies the desired conditions. Suppose $i_1, \ldots, i_r$ are the copies of node $i$ created in this process; then the final value of $\sum_{l=1}^{r} y_{i_l}$ is the same as the initial value of $y_i$.

We now proceed with the proof. For a demand node $j$, let $\sigma(j)$ be the facility assigned to $j$ in the current solution. With every facility $i$, we will associate a modified solution as follows. Let $D_{SOL}(i)$ be the set of all demand nodes $j$ such that $x_{ij} > 0$. Note that $x_{ij} = y_i$ for all $i \in D_{SOL}(i)$. Consider the solution obtained by including facility $i$ in the current solution and reassigning all nodes in $D_{SOL}(i)$ to $i$. Let gain$'(i)$ be the decrease in cost of the solution as a result of this modification, i.e.,

$$\text{gain}'(i) = -f_i + \sum_{j \in D_{SOL}(i)} (c_{\sigma(j)j} - c_{ij}).$$

Note that gain$'(i)$ could be $< 0$. Clearly, $\text{gain}(i) \geq \text{gain}'(i)$. We will prove that $\sum_i y_i \cdot \text{gain}'(i) = C - (F_{SOL} + C_{SOL})$. Since $y_i = x_{ij}$ if $j \in D_{SOL}(i)$ (also using the fact that $x_{ij} = 0$ if $j \notin D_{SOL}(i)$ to simplify the index of the summation to $j$), we get

$$y_i \cdot \text{gain}'(i) = -y_i f_i + \sum_j x_{ij} c_{\sigma(j)j} - \sum_j x_{ij} c_{ij}.$$

Summing over all $i \in \mathcal{F}$, the first term evaluates to $-F_{SOL}$ and the third term to $-C_{SOL}$. The second term, if we reverse the order of the summation indices, using $\sum_i x_{ij} = 1$, simplifies to $\sum_j c_{\sigma(j)j}$, which evaluates to $C$, which proves $\sum \text{gain}(i) \geq -F_{SOL} + C - C_{SOL}$. □

LEMMA 2.7. *$\sum \text{gain}(i) \geq F - (F_{SOL} + 2C_{SOL})$, where $F_{SOL}$ and $C_{SOL}$ are the facility and service costs for an arbitrary fractional solution SOL to the facility location LP.*

*Proof.* Consider the fractional solution $SOL$ to the facility location LP. Let $y_i$ denote the variable that indicates whether facility $i$ is chosen in the solution and let $x_{ij}$ be the variable that indicates whether node $j$ is serviced by facility $i$. $F_{SOL} = \sum_i y_i f_i$ and $C_{SOL} = \sum_{ij} x_{ij} c_{ij}$. We will prove that $\sum_i y_i \cdot \text{gain}(i) \geq F - (F_{SOL} + 2C_{SOL})$. As before, assume without loss of generality that $y_i \leq 1$ for all $i$ and $\sum_i x_{ij} = 1$ for all $ij$.

Let $\mathcal{F}$ be the set of facilities in the current solution. Mimicking the proof of Lemma 2.5, we will match each node $i' \in \mathcal{F}$ to its "nearest" node in the fractional solution. However, since we have a fractional solution, this matching is a *fractional matching*, given by variables $m_{ii'} \geq 0$, where the value of $m_{ii'}$ indicates the extent to which $i'$ is matched to $i$. The variables satisfy the constraints $m_{ii'} \leq y_i$, $\sum_i m_{ii'} = 1$, and the values are chosen so as to minimize $\sum_i m_{ii'} c_{ii'}$. So for any $j$,

$$\sum_i m_{ii'} c_{ii'} \leq \sum_i x_{ij} c_{ii'} \leq \sum_i x_{ij}(c_{i'j} + c_{ij})$$
$$\leq c_{i'j} + \sum_i x_{ij} c_{ij}.$$

In particular, for $i' = \sigma(j)$, we get

$$\sum_i m_{i\sigma(j)} c_{i\sigma(j)} \leq c_{\sigma(j)j} + \sum_i x_{ij} c_{ij}. \tag{6.1}$$

As in the proof of Lemma 2.5, we modify the fractional solution by making multiple copies of facilities such that for every $ij$, either $x_{ij} = 0$ or $x_{ij} = y_i$ and additionally, for every $i, i'$, either $m_{ii'} = 0$ or $m_{ii'} = y_i$. (The second condition is enforced in exactly the same way as the first, by treating the variables $m_{ii'}$ just as the variables $x_{ij}$.)

For a demand node $j$, let $\sigma(j)$ be the facility assigned to $j$ in the current solution. Let $D(i')$ be the set of demand nodes $j$ assigned to facility $i'$ in the current solution. With every facility $i$, we will associate a modified solution as follows. Let $D_{SOL}(i)$ be the set of all demand nodes $j$ such that $x_{ij} > 0$. Let $R(i)$ be the set of facilities $i' \in \mathcal{F}$ such that $m_{ii'} > 0$. Note that $x_{ij} = y_i$ for all $j \in D_{SOL}(i)$ and $m_{ii'} = y_i$ for all $i' \in R(i)$. Consider the solution obtained by including facility $i$ in the current solution and reassigning all nodes in $D_{SOL}(i)$ to $i$. Further, for all facilities $i' \in R(i)$, the facility $i'$ is removed from the solution and all nodes in $D(i') \setminus D_{SOL}(i)$ are reassigned to $i$. Let gain$'(i)$ be the decrease in cost of the solution as a result of this modification, i.e.,

$$\text{gain}'(i) = -f_i + \sum_{j \in D_{SOL}(i)} (c_{\sigma(j)j} - c_{ij}) + \sum_{i' \in R(i)} \left( f_{i'} + \sum_{j \in D(i') \setminus D_{SOL}(i)} (c_{\sigma(j)j} - c_{ij}) \right).$$

Clearly, gain$(i) \geq$ gain$'(i)$. We will prove that $\sum_i y_i \cdot \text{gain}'(i) \geq F - (F_{SOL} + 2C_{SOL})$. We also know that if $i' \in R(i)$, then $m_{ii'} = y_i$. Since $m_{ii'} = 0$ if $i'$ is not in $R(i)$, we will drop the restriction on $i'$ in the summation:

$$y_i \cdot \text{gain}'(i) = -y_i f_i + \sum_{j \in D_{SOL}(i)} x_{ij}(c_{\sigma(j)j} - c_{ij})$$
$$+ \sum_{i'} m_{ii'} \left( f_{i'} + \sum_{j \in D(i') \setminus D_{SOL}(i)} (c_{\sigma(j)j} - c_{ij}) \right).$$

From triangle inequality we have $-c_{i'i} \leq c_{i'j} - c_{ij}$. We also replace $i'$ by $\sigma(j)$ wherever convenient. Therefore we conclude that

$$y_i \cdot \text{gain}'(i) \geq -y_i f_i + \sum_{j \in D_{SOL}(i)} x_{ij}(c_{\sigma(j)j} - c_{ij})$$
$$+ \sum_{i'} \left( m_{ii'} f_{i'} + \sum_{j \in D(i') \setminus D_{SOL}(i)} -m_{i\sigma(j)} c_{i\sigma(j)} \right).$$

Once again evaluating the negative terms in the last summand over a larger set (namely, all $i', j \in D(i')$ instead of $i', j \in D(i') \setminus D_{SOL}(i)$; but since the $D(i')$'s are disjoint this simplifies to a sum over all $j$) and summing the result over all $i$, we have

$$\sum_i y_i \cdot \text{gain}'(i) \geq -\sum_i y_i \cdot f_i + \sum_i \sum_{j \in D_{SOL}(i)} x_{ij}(c_{\sigma(j)j} - c_{ij}) + \sum_i \sum_{i'} m_{ii'} f_{i'} - \sum_i \sum_j m_{i\sigma(j)} c_{\sigma(j)i}.$$

The first term in the above expression is equal to $-F_{SOL}$. The second term has two parts, the latter of which is $\sum_{i,j} x_{ij}c_{ij}$, which evaluates to the fractional service cost, $C_{SOL}$. The first part of the second term evaluates to $\sum_j c_{\sigma(j)j}$ since, if we reverse the order of the summation, $\sum_{i,j \in D_{SOL}(i)} x_{ij} = 1$, since node $j$ is assigned fractionally. This part evaluates to $C$.

The third term is equal to $\sum_{i'} f_{i'}$, again reversing the order of the summation and using the fact that $\sum_i m_{ii'} = 1$ from the fractional matching of the nodes $i'$ in the current solution.

We now bound the last term in the expression. Notice the sets $R(i)$ may not be disjoint and we require a slightly different approach than that in the proof of Lemma 2.5. We use the inequality (6.1), and the term (which is now $-\sum_i \sum_j m_{i\sigma(j)}c_{\sigma(j)i}$) is at least $-\sum_j c_{\sigma(j)j} - \sum_{i,j} x_{ij}c_{ij}$. The first part evaluates to $-C$ and the second part to $-C_{SOL}$. Substituting these expressions, we get

$$\sum_i y_i \cdot \text{gain}'(i) \geq -F_{SOL} + F + C - C_{SOL} + F - C - C_{SOL},$$

which proves the lemma. □

**Implementing local search.** We ran some preliminary experiments with an implementation of the basic local search algorithm as described in section 2. We tested the program on the data sets at the OR-Library (http://people.brunel.ac.uk/~mastjjb/jeb/info.html). The results were very encouraging, and the program found the optimal solution for 10 of the 15 data sets. One set had 3.5% error, and the others had less than 1% error. The program took less than 10 seconds (on a PC) to run in all cases. This study is by no means complete, since we did not implement scaling, primal-dual algorithms, and the other well-known heuristics in the literature.

The heuristic deleted three nodes a few times and two nodes several times upon insertion of a new facility. Thus it seems that the generalization of deleting more than one facility (cf. add and drop heuristics; see Kuehn and Hamburger [24] and Korupolu, Plaxton, and Rajaraman [23]) was useful in the context of these data sets.

## 7. Concurrent and subsequent results.

Subsequent to the publication of the extended abstract of this paper [7], several results have been published regarding the facility location problem. The result in [27] provides a 1.86 approximation while running in time $O(m \log m)$ in a graph with $m$ edges using dual fitting. [36] also proves results in a similar vein. [20] gives a 1.61 approximation for the uncapacitated facility location problem, which is improved to 1.52 in [28]. [28] also gives a 2.88 approximation for the soft capacitated problem discussed in this paper.

Although this paper concentrates on the facility location problem, two results on the related $k$-median problem are of interest to the techniques considered here. The first is the result of Mettu and Plaxton [30], which gives an $O(1)$ approximation to the $k$-median problem, using combinatorial techniques. They output a list of points such that for any $k$, the first $k$ points in the output list give an $O(1)$ approximation to the $k$-median problem.

The second result is of Arya et al. in [2] where the authors present a $3 + \epsilon$ approximation algorithm for the $k$-median problem based on local search. They show how to use exchanges as opposed to the general delete procedure considered here to prove an approximation bound.

**Acknowledgments.** We would like to thank Chandra Chekuri, Ashish Goel, Kamal Jain, Samir Khuller, Rajeev Motwani, David Shmoys, Éva Tardos, and Vijay Vazirani for numerous discussions on these problems. We would also like to thank the referees for helping us improve the presentation of this manuscript significantly.

## REFERENCES

[1] S. Arora, P. Raghavan, and S. Rao, *Approximation schemes for Euclidean k-medians and related problems*, in Proceedings of the 30th Annual ACM Symposium on Theory of Computing, 1998, pp. 106–113.

[2] V. Arya, N. Garg, R. Khandekar, A. Meyerson, K. Munagala, and V. Pandit, *Local search heuristics for k-median and facility location problems*, SIAM J. Comput. 33, (2004), pp. 544–562.

[3] J. Bar-Ilan, G. Kortsarz, and D. Peleg, *How to allocate network centers*, J. Algorithms, 15 (1993), pp. 385–415.

[4] Y. Bartal, *Probabilistic approximation of metric spaces and its algorithmic applications*, in Proceedings of the 37th Annual IEEE Symposium on Foundations of Computer Science, 1996, pp. 184–193.

[5] Y. Bartal, *On approximating arbitrary metrics by tree metrics*, in Proceedings of the 30th Annual ACM Symposium on Theory of Computing, 1998, pp. 161–168.

[6] M. Charikar, C. Chekuri, A. Goel, and S. Guha, *Rounding via trees: Deterministic approximation algorithms for group Steiner trees and k-median*, in Proceedings of the 30th Annual ACM Symposium on Theory of Computing, 1998, pp. 114–123.

[7] M. Charikar and S. Guha, *Improved combinatorial algorithms for the facility location and k-median problems*, in Proceedings of the 39th Annual IEEE Symposium on Foundations of Computer Science, 1999, pp. 378–388.

[8] M. Charikar, S. Guha, É. Tardos, and D. B. Shmoys, *A constant factor approximation algorithm for the k-median problem*, J. Comput. System Sci., 65 (2002), pp. 129–149.

[9] F. A. Chudak, *Improved approximation algorithms for uncapacitated facility location*, in Integer Programming and Combinatorial Optimization, R. E. Bixby, E. A. Boyd, and R. Z. Ríos-Mercado, eds., Lecture Notes in Comput. Sci. 1412, Springer-Verlag, Berlin, 1998, pp. 180–194.

[10] F. A. Chudak and D. B. Shmoys, *Improved approximation algorithms for the uncapacitated facility location problem*, SIAM J. Comput., 33 (2003), pp. 1–25.

[11] F. A. Chudak and D. B. Shmoys, *Improved approximation algorithms for a capacitated facility location problem*, in Proceedings of the 10th Annual ACM-SIAM Symposium on Discrete Algorithms, 1999, pp. S875–S876.

[12] F. A. Chudak and D. P. Williamson, *Improved approximation algorithms for capacitated facility location problems*, in Proceedings of Integer Programming and Combinatorial Optimization, Lecture Notes in Comput. Sci. 1610, Springer-Verlag, Berlin, 1999, pp. 99–113.

[13] G. Cornuéjols, G. L. Nemhauser, and L. A. Wolsey, *The uncapacitated facility location problem*, in Discrete Location Theory, P. Mirchandani and R. Francis, eds., John Wiley and Sons, New York, 1990, pp. 119–171.

[14] M. E. Dyer and A. M. Frieze, *A simple heuristic for the p-center problem*, Oper. Res. Lett., 3 (1985), pp. 285–288.

[15] T. F. Gonzalez, *Clustering to minimize the maximum intercluster distance*, Theoret. Comput. Sci., 38 (1985), pp. 293–306.

[16] S. Guha and S. Khuller, *Greedy strikes back: Improved facility location algorithms*, J. Algorithms, 31 (1999), pp. 228–248.

[17] D. S. Hochbaum and D. B. Shmoys, *A best possible approximation algorithm for the k-center problem*, Math. Oper. Res., 10 (1985), pp. 180–184.

[18] P. Indyk, *Sublinear time algorithms for metric space problems*, in Proceedings of the 31st Annual ACM Symposium on Theory of Computing, 1999, pp. 428–434.

[19] K. Jain and V. Vazirani, *Approximation algorithms for metric facility location and k-median problems using the primal-dual schema and Lagrangian relaxation*, J. ACM, 48 (2001), pp. 274–296.

[20] K. Jain, M. Mahdian, and A. Saberi, *A new greedy approach for facility location problems*, in Proceedings of the 34th Annual ACM Symposium on Theory of Computing, 2002, pp. 731–740.

[21] O. Kariv and S. L. Hakimi, *An algorithmic approach to network location problems.* II: *The p-medians*, SIAM J. Appl. Math., 37 (1979), pp. 539–560.

[22] S. Khuller and Y. J. Sussmann, *The capacitated $K$-center problem*, SIAM J. Discrete Math., 13 (2000), pp. 403–418.

[23] M. R. Korupolu, C. G. Plaxton, and R. Rajaraman, *Analysis of a local search heuristic for facility location problems*, J. Algorithms, 37 (2000), pp. 146–188.

[24] A. A. Kuehn and M. J. Hamburger, *A heuristic program for locating warehouses*, Management Science, 9 (1979), pp. 643–666.

[25] J.-H. Lin and J. S. Vitter, *Approximation algorithms for geometric median problems*, Inform. Proc. Lett., 44 (1992), pp. 245–249.

[26] J.-H. Lin and J. S. Vitter, *$\epsilon$-approximations with minimum packing constraint violation*, in Proceedings of the 24th Annual ACM Symposium on Theory of Computing, 1992, pp. 771–782.

[27] K. Jain, M. Mahdian, E. Markakis, A. Saberi, and V. V. Vazirani, *Greedy facility location algorithms analyzed using dual fitting with factor-revealing LP*, J. ACM, 50 (2003), pp. 795–824.

[28] M. Mahdian, Y. Ye, and J. Zhang, *Improved approximation algorithms for metric facility location problems*, in Proceedings of Approximation Algorithms for Combinatorial Optimization, Lecture Notes in Comput. Sci. 2462, Springer-Verlag, Berlin, 2002, pp. 229–242.

[29] M. V. Marathe, R. Ravi, R. Sundaram, S. S. Ravi, D. J. Rosenkrantz, and H. B. Hunt, III, *Bicriteria network design problems*, J. Algorithms, 28 (1998), pp. 142–171.

[30] R. R. Mettu and C. G. Plaxton, *The online median problem*, SIAM J. Comput., 32 (2003), pp. 816–832.

[31] D. B. Shmoys, *personal communication*, 1999.

[32] D. B. Shmoys and É. Tardos, *An approximation algorithm for the generalized assignment problem*, Math. Program., 62 (1993), pp. 461–474.

[33] D. B. Shmoys, É. Tardos, and K. I. Aardal, *Approximation algorithms for facility location problems*, in Proceedings of the 29th Annual ACM Symposium on Theory of Computing, 1997, pp. 265–274.

[34] M. Sviridenko, *personal communication*, 1998.

[35] A. Tamir, *An $O(pn^2)$ algorithm for the p-median and related problems on tree graphs*, Oper. Res. Lett., 19 (1996), pp. 59–94.

[36] M. Thorup, *Quick k-median, k-center, and facility location for sparse graphs*, in Proceedings of ICALP, Lecture Notes in Comput. Sci. 2076, Springer-Verlag, Berlin, 2001, pp. 249–260.

[37] S. de Vries, M. Posner, and R. Vohra, *The $K$-Median Problem on a Tree*, working paper, Ohio State University, Columbus, OH, 1998.

[38] J. Ward, R. T. Wong, P. Lemke, and A. Oudjit, *Properties of the Tree $K$-Median Linear Programming Relaxation*, Research report CC-878-29, Institute for Interdisciplinary Engineering Studies, Purdue University, West Lafayette, IN, 1987.

SIAM J. COMPUT.
Vol. 34, No. 4, pp. 825–847

# ON THE RELATIONSHIP BETWEEN CLIQUE-WIDTH AND TREEWIDTH*

DEREK G. CORNEIL[†] AND UDI ROTICS[‡]

**Abstract.** Treewidth is generally regarded as one of the most useful parameterizations of a graph's construction. Clique-width is a similar parameterization that shares one of the powerful properties of treewidth, namely: if a graph is of bounded treewidth (or clique-width), then there is a polynomial time algorithm for any graph problem expressible in monadic second order logic, using quantifiers on vertices (in the case of clique-width you must assume a clique-width parse expression is given). In studying the relationship between treewidth and clique-width, Courcelle and Olariu [*Discrete Appl. Math.*, 101 (2000), pp. 77–114] showed that any graph of bounded treewidth is also of bounded clique-width; in particular, for any graph $G$ with treewidth $k$, the clique-width of $G$ is at most $4 * 2^{k-1} + 1$.

In this paper, we improve this result by showing that the clique-width of $G$ is at most $3 * 2^{k-1}$ and, more importantly, that there is an exponential lower bound on this relationship. In particular, for any $k$, there is a graph $G$ with treewidth equal to $k$, where the clique-width of $G$ is at least $2^{\lfloor k/2 \rfloor - 1}$.

**Key words.** clique-width, treewidth

**AMS subject classifications.** 05C75, 05C78, 05C85

**DOI.** 10.1137/S0097539701385351

## 1. Introduction.

One of the most fruitful graph theoretical developments of the last few decades has been the concept of treewidth, pioneered by Robertson and Seymour. Loosely speaking, the treewidth of a graph captures a way of constructing the graph in a "tree-like" fashion. The lower the treewidth of a graph, the closer it is to being a tree (connected treewidth one graphs are precisely trees). One of the major results in this area is that any problem expressible in monadic second order logic (which includes many NP-complete graph problems), when restricted to graphs of bounded treewidth $k$, has a linear time algorithm (albeit with a constant that grows exponentially with $k$). Although this result is very far reaching, it is still somewhat dissatisfying since many classes of "tame" graphs, for example, cliques, have arbitrarily high treewidth, yet have simple linear time algorithms for most of the problems mentioned above.

The clique-width of a graph is another attempt to parameterize the construction of a graph so that sweeping claims can be made about the graph's tractability for polynomial time solutions to difficult problems. The clique-width of a graph $G$, denoted by $cwd(G)$, is defined as the minimum number of labels needed to construct $G$, using the following four graph operations: creation of a new vertex $v$ with label $i$ (denoted $i(v)$), disjoint union ($\oplus$), connecting vertices with specified labels ($\eta$), and renaming labels ($\rho$). The construction of a graph $G$ using the above four operations

*Received by the editors February 16, 2001; accepted for publication (in revised form) August 17, 2004; published electronically April 29, 2005. An extended abstract of this paper appeared in *Proceedings of the 27th International Workshop on Graph-Theoretic Concepts in Computer Science,* 2001, Lecture Notes in Comput. Sci. 2204, Springer-Verlag, Berlin, 2001, pp. 78–90.
http://www.siam.org/journals/sicomp/34-4/38535.html

[†]Department of Computer Science, University of Toronto, Toronto M5S 3G4, ON, Canada (dgc@cs.toronto.edu).

[‡]School of Mathematics and Computer Science, Netanya Academic College, Netanya, Israel (rotics@mars.netanya.ac.il).

is represented by an algebraic expression called a *$k$-expression*, where $k$ is the number of labels used in the expression. More details are given in section 2. This notion was first introduced by Courcelle, Engelfriet, and Rozenberg in [5] and has been studied extensively in recent years.

For example, cographs (graphs with no induced $P_4$'s) are exactly the graphs of clique-width at most two, and trees have clique-width at most three [8]. $P_4$-sparse (every five vertices have at most one $P_4$) and $P_4$-tidy graphs (no induced $P_4$ has more than one partner, where a partner is a vertex whose inclusion in the $P_4$ results in at least two distinct $P_4$'s) have clique-width at most four [6]. The $(q, q-4)$ graphs for $q$ at least four and $(q, q-3)$ graphs for $q$ at least seven have clique-width at most $q$ [6, 12]. A $(q, t)$ graph is a graph in which every subgraph induced by $q$ vertices contains at most $t$ induced $P_4$'s.

As mentioned above, the motivation for studying clique-width is analogous to that of treewidth. In particular, problems defined by monadic second order logic formulas, using quantifiers on vertices but not on edges, can be solved in polynomial time on any class of graphs $\mathcal{C}$ of clique-width at most $k$, for some fixed $k$, assuming that the $k$-expression defining the input graph is given. For details, see [6, 7]. In addition, polynomial time algorithms can be obtained for other problems such as chromatic number, edge dominating set [11], and $ID_q$-partition problems [10], on any class of graphs $\mathcal{C}$ of clique-width at most $k$, for some fixed $k$, assuming that the $k$-expression defining the input graph is given.

This raises the obvious question about the relationship between treewidth and clique-width. Courcelle and Olariu [8] proved the following theorem.

THEOREM 1.1 (see [8]). *If the treewidth of $G$ is $k$, then $cwd(G) \leq 2^{k+1} + 1$ $(= 4 * 2^{k-1} + 1)$.*

This theorem guarantees that any class of graphs of bounded treewidth is also of bounded clique-width and thus, from the perspective of algorithmic tractability, clique-width is a more powerful concept. Note that since the clique-width of a clique on $n$ vertices is at most two and its treewidth is $n-1$, the gap between a graph's treewidth and clique-width can be arbitrarily high. The above theorem also raises the questions of whether the bound can be improved (note that for trees, $k$ equals one, and the theorem guarantees that the clique-width of a tree is at most five, whereas it is known that the clique-width of a tree is at most three) and more importantly, whether the exponential bound represented in the theorem can be replaced by a polynomial bound. To make this more precise, we let $\mathcal{G}_k = \{G : twd(G) = k\}$, where $twd(G)$ denotes the treewidth of $G$, and want to determine whether $f(\mathcal{G}_k) = max\{cwd(G) : G \in \mathcal{G}_k\}$ can grow polynomially with $k$.

In answer to these two questions, our paper proves the following two theorems.

THEOREM 1.2. *If the treewidth of $G$ is $k$, then $cwd(G) \leq 2^k + 2^{k-1}$ $(= 3 * 2^{k-1})$.*

THEOREM 1.3. *For any $k$, there is a graph $G$ where $twd(G) = k$ and $cwd(G) \geq 2^{\lfloor k/2 \rfloor - 1}$ (i.e., $f(\mathcal{G}_k) \geq 2^{\lfloor k/2 \rfloor - 1}$).*

*Overview of the paper.* The second section of the paper introduces the notation and definitions used throughout the paper. In particular, we present the concept of $k$-trees and partial $k$-trees. (Note that partial $k$-trees encompass an alternative definition of graphs having treewidth at most $k$.) We also describe how to construct the $k$-trees that will be used in the proofs of the two theorems. Sections 3 and 4 are devoted to the proofs of Theorems 1.2 and 1.3, respectively. The paper ends with concluding remarks and directions for further research.

## 2. Notation and definitions.

The graphs we consider in this paper are undirected and loop-free. For a graph $G$ and a vertex $u$ of $G$, we denote by $N_G(u)$ the

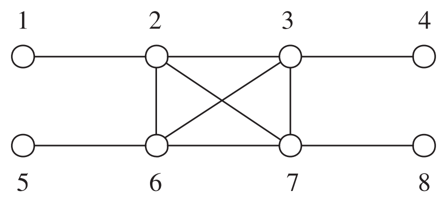


FIG. 1. *A graph $G$ of clique-width three.*

$$\begin{aligned}\mathbf{t_1} &= \boldsymbol{\rho}_{\mathbf{c}\to\mathbf{b}}(\boldsymbol{\eta}_{\mathbf{b,c}}(\boldsymbol{\eta}_{\mathbf{a,b}}(\mathbf{a(1)} \oplus \mathbf{b(2)}) \oplus \boldsymbol{\eta}_{\mathbf{a,c}}(\mathbf{a(5)} \oplus \mathbf{c(6)})))\\ \mathbf{t_2} &= \boldsymbol{\rho}_{\mathbf{b}\to\mathbf{c}}(\boldsymbol{\eta}_{\mathbf{b,c}}\,(\boldsymbol{\eta}_{\mathbf{a,b}}(\mathbf{a(4)} \oplus \mathbf{b(3)}) \oplus \boldsymbol{\eta}_{\mathbf{a,c}}(\mathbf{a(8)} \oplus \mathbf{c(7)})))\\ \mathbf{t} &= \boldsymbol{\eta}_{\mathbf{b}\to\mathbf{c}}(\mathbf{t_1} \oplus \mathbf{t_2})\end{aligned}$$

FIG. 2. *A 3-expression defining the graph of Figure 1.*

*neighborhood* of $u$ in $G$, which is the set of all vertices in $G$ that are adjacent to $u$. We denote by $N_G[u]$ the *closed neighborhood* of $u$ in $G$ which is equal to $N_G(u) \cup \{u\}$.

We first give more details on the definition of clique-width presented above. The *clique-width* of a graph $G$, denoted by $cwd(G)$, is defined as the minimum number of labels needed to construct $G$, using the following four graph operations: creation of a new vertex $v$ with label $i$ (denoted $i(v)$), disjoint union ($\oplus$), connecting vertices with specified labels ($\eta$), and renaming labels ($\rho$). The operation $\eta_{i,j}$ ($i \neq j$) adds all edges (that are not already present) between every vertex of label $i$ and every vertex of label $j$. The operation $\rho_{i\to j}$ renames all vertices of label $i$ with label $j$. An expression built from the above four operations is called a *clique-width expression.* A clique-width expression using $k$ labels is called a *$k$-expression.* Each $k$-expression $t$ uniquely defines a labeled graph $val(t)$, where the labels are integers $1, \dots, k$ associated with the vertices and each vertex has exactly one label. We say that a $k$-expression $t$ defines a graph $G$ if $G$ is equal to the graph obtained from the labeled graph $val(t)$ after removing its labels. The clique-width of a graph $G$ is equal to the minimum $k$ such that there exists a $k$-expression defining $G$.

For a $k$-expression $t$ defining a graph $G$, we denote by $tree(t)$ the parse tree constructed from $t$ in the usual way. The leaves of this tree are the vertices of $G$ with their initial labels, and the internal nodes correspond to the operations of $t$ and can be either binary corresponding to $\oplus$, or unary corresponding to $\eta$ or $\rho$. If $k$ equals $cwd(G)$, we say that $k$-expression $t$ is an *optimum clique-width expression* of $G$ and we say that $tree(t)$ is an *optimum clique-width parse tree* for $G$. For a vertex $u$ of $G$ and an internal node $a$ of $tree(t)$, such that $u$ occurs (as a leaf) in the subtree of $tree(t)$ rooted at $a$, the *label of $u$ at $a$* is defined as the label that $u$ has immediately before the operation $a$ is applied on the subtree of $tree(t)$ rooted at $a$.

*Example* 2.1. Figure 1 illustrates a graph $G$ of clique-width three. The 3-expression $t$ presented in Figure 2 defines the graph $G$ of Figure 1. Note that the labeled graph $val(t)$ can be obtained from the graph $G$ by adding label $a$ to vertices 1, 4, 5, and 8, label $b$ to vertices 2 and 3, and label $c$ to vertices 6 and 7. The 3-expression $t$ shows that $G$ is of clique-width at most three. Since $G$ has an induced $P_4$ and cographs (the graphs with no induced $P_4$'s) are exactly the graphs of clique-width at most two [8], it follows that the clique-width of $G$ is exactly three. Figure 3 presents the optimum clique-width parse tree, denoted $tree(t)$, that corresponds

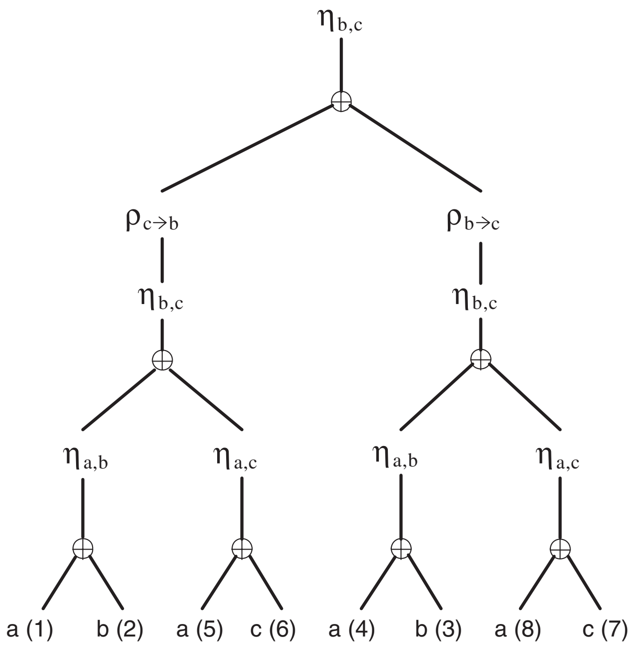


FIG. 3. *The optimum clique-width parse tree for the graph of Figure* 1 *corresponding to the* 3-*expression of Figure* 2.

to the 3-expression $t$ presented in Figure 2.

As mentioned above, $G$ being a partial $k$-tree is equivalent to $G$ having treewidth at most $k$. We now define $k$-trees and present notation for the $k$-tree that will be used to establish the upper and lower bounds presented in sections 3 and 4.

We denote by $K_k$ the clique with $k$ vertices. For a vertex $v$ and a set of vertices $S$ we say that $v$ is *universal* to $S$ if $v$ is adjacent to all the vertices in $S$. A graph $G$ is a *$k$-tree* if $G$ is either a $K_k$ or is formed from a $k$-tree $G'$, by adding a new vertex that is universal to a $K_k$ in $G'$. Thus, a $k$-tree $G$ can be constructed by taking an initial $K_k$ of $G$ and adding the other vertices of $G$ one after the other. At each step a new vertex $x$ is added to $G$ that is universal to a $k$-clique (denoted by $Q_x$) in $G$, consisting of $k$ vertices that were added to $G$ before $x$.

Note that 1-trees are precisely trees. $G$ is a *partial $k$-tree* (i.e., $twd(G) \leq k$) if $G$ is a partial subgraph of a $k$-tree. This recursive definition of $k$-trees immediately suggests that the construction of a given $k$-tree $G$ can be represented by a tree $T_G$ as defined below.

We let the initial $K_k$ be on the vertices $\{1, 2, \ldots, k\}$ and assume that the rest of the vertices of $G$ are numbered $k+1, k+2, \ldots, n$ in the order in which they were added to $G$. The nodes of the tree $T_G$ correspond to vertices $k, k+1, \ldots, n$ of $G$, where each node $i$ of $T_G$ corresponds to vertex $i$ of $G$. The root of $T_G$, denoted by $R$, corresponds to vertex $k$ of $G$. As mentioned above, when a vertex $i$ is added to the graph $G$, it is made universal to $Q_i$, a $k$-clique that consists of $k$ vertices that were added to $G$ before $i$. We denote by $p(i)$ the largest (according to the numbering of the vertices) vertex in $Q_i$. The tree $T_G$ is constructed by making each node $i$, $i > k$, of $T_G$, a child of $p(i)$.

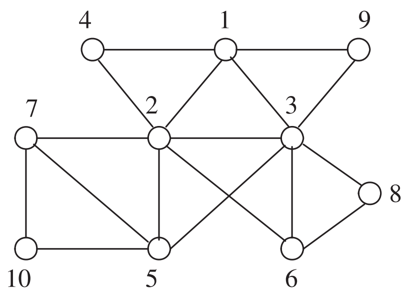


FIG. 4. *A 2-tree.*

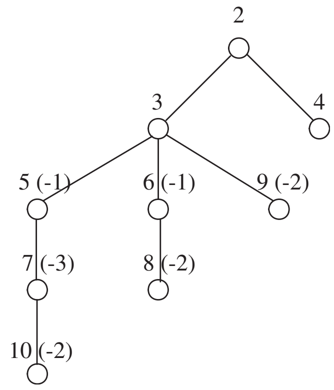


FIG. 5. *The construction tree for the graph of Figure 4.*

If $i$ is not a child of the root $R$, then $Q_i$ can be obtained from $Q_{p(i)}$ by removing vertex $j$ for some $j$ in $Q_{p(i)}$, $1 \leq j < p(i)$, and replacing it with vertex $p(i)$. In this case we will say that node $i$ is a $-j$ child of node $p(i)$ in the tree $T_G$. (Figure 4 contains an example of a 2-tree, and Figure 5 contains its construction tree.) If $i$ is a child of $R$, then $Q_i$ must be equal to $\{1, 2, \ldots, k\}$. Note that for the root $R$, $Q_R$ is not defined.

For a node $i$ of $T_G$ other than the root $R$, we call the $k-1$ vertices of $Q_i - \{p(i)\}$ the *sources* of $i$. Clearly, the sources of $i$ that are not in $\{1, 2, \ldots, k\}$ are ancestors of $i$ in the tree $T_G$.

## 3. Upper bound.

We now turn our attention to establishing the upper bound stated in Theorem 1.2.

THEOREM 1.2. *If the treewidth of $G$ is $k$, then $cwd(G) \leq 2^k + 2^{k-1}$* $(= 3 * 2^{k-1})$.

The bound is achieved for a $k$-tree, and in our proof, we will first prove it for $G$ a $k$-tree and then indicate the modifications for partial $k$-trees. We present an algorithm that is guaranteed to generate a clique-width expression that uses at most $3 * 2^{k-1}$ labels. It should be clear to the reader that this algorithm is an extension of the standard algorithm that shows that the clique-width of a tree (i.e., a 1-tree) is at most three.

Given $G$, let $T_G$ be a construction tree for it. For $x$, a node in $T_G$, we let $T_x$ denote the subtree of $T_G$ rooted at $x$. In the algorithm we will use the following two "metaoperations":

- Let $L = \{l_1, l_2, \ldots, l_h\}$ be a set of labels and let $l^*$ be another label. Then $\eta_{l^*,L}$

denotes the following sequence of $\eta$ operations: $\eta_{l^*,l_1}\,\eta_{l^*,l_2}\cdots\eta_{l^*,l_h}$. (Note that since there are no $\rho$ or $\oplus$ operations in the sequence, the order of these $\eta$ operations is not important.)

- Let $P = \{(l_1,l'_1),(l_2,l'_2),\dots,(l_h,l'_h)\}$ be a set of pairs of labels such that no label in $\{l_1,l_2,\dots,l_h\}$ is equal to a label in $\{l'_1,l'_2,\dots,l'_h\}$ and $l_i \neq l_j$ for $i \neq j$. Then $\rho_P$ denotes the following sequence of $\rho$ operations: $\rho_{l_1\to l'_1}\,\rho_{l_2\to l'_2}\cdots\rho_{l_h\to l'_h}$. (Note the requirements that no label in $\{l_1,l_2,\dots,l_h\}$ is equal to a label in $\{l'_1,l'_2,\dots,l'_h\}$ and $l_i \neq l_j$ imply that the order of these $\rho$ operations is not important.)

ALGORITHM 1.

**Overview.** The algorithm will proceed by bottom-up dynamic programming on $T_G$. Let $x$ be an arbitrary nonroot vertex in $T_G$ where the sources of $x$ are $\{s_1,s_2,\dots,s_{k-1}\}$ and $p(x)$ is $s_k$. Each child of $x$ will be a $-s_i$ node for some $i$ in $\{1,2,\dots,k\}$ (i.e., such a node will be universal to the $k$-clique consisting of $\{s_1,s_2,\dots,s_k\}-\{s_i\}\cup\{x\}$). Note that the subtree rooted at a $-s_i$ child of $x$ could contain vertices universal to any subset of $\{s_1,s_2,\dots,s_k\}-\{s_i\}\cup\{x\}$. When processing $x$ it will be assumed that the subtrees rooted at children of $x$ have a corresponding clique-width expression whose generation required at most $3*2^{k-1}$ labels and that each vertex in the subtree rooted at a $-s_i$ child of $x$ is labeled with one of $2^k$ labels indicating the subset of $\{s_1,s_2,\dots,s_k\}-\{s_i\}\cup\{x\}$ to which it is universal (clearly this assumption holds for the leaves of $T_G$).

The algorithm, using at most $3*2^{k-1}$ labels, will construct the clique-width expression $t_x$ for $T_x$ so that each vertex in $T_x$ is assigned one of $2^k$ labels indicating the subset of $\{s_1,s_2,\dots,s_k\}$ to which it is adjacent. For $1\le i\le k$, let $y_i$ denote a $-s_i$ child of $x$. Let set $\mathcal{T}_i$ be $\{T_{y_i} : y_i \text{ is a } -s_i \text{ child of } x\}$. All vertices in $\mathcal{T}_i$ adjacent to the same subset of $\{s_1,s_2,\dots,s_k\}-\{s_i\}\cup\{x\}$ will be assumed to have the same label. In particular, we let $t_i$ denote the clique-width expression for $\mathcal{T}_i$, constructed by the algorithm. We assume that the graph $val(t_i)$ has $2^k$ labels, each label corresponding to a subset of $\{s_1,\dots,s_k\}-\{s_i\}\cup\{x\}$. For each vertex $w$ in $val(t_i)$, its label indicates the set of vertices in $\{s_1,\dots,s_k\}-\{s_i\}\cup\{x\}$ to which it is adjacent. The labeling of the vertices in $val(t_1), val(t_2),\dots,val(t_k)$ is unique in the sense that the same label will be used in the various $val(t_i)$ to indicate a particular subset of $\{s_1,\dots,s_k\}\cup\{x\}$.

We now describe the steps of the algorithm. Steps 1 and 2 create $t_x$, the clique-width expression of $T_x$, for $x$ a nonroot vertex. Step 3 processes the root.

**Step 1.** Let $X$ be an arbitrary subset of $\{s_2,\dots,s_k\}$ and let $l_X$ (respectively, $l_{X\cup\{x\}}$) denote the label corresponding to $X$ (respectively, $X\cup\{x\}$). The set $\{l_{X\cup\{x\}} : X\subseteq\{s_2,\dots,s_k\}\}$ will be represented by $L_1$, and the set of pairs $\{(l_{X\cup\{x\}}, l_X) : X\subseteq\{s_2,\dots,s_k\}\}$ will be represented by $P_1$. The following operation will form the new clique-width expression $t'_1$ that introduces vertex $x$ where vertex $x$ is assigned a label (denoted $l^*$) that corresponds to the set $\{s_1,\dots,s_k\}$; note that since $x$ is the only vertex in $T_x$ adjacent to $\{s_1,s_2,\dots,s_k\}$, its label will not appear in any $t_i$, $1\le i\le k$. Recall that $t_1$ is the clique-width expression for $\mathcal{T}_1$:

$$t'_1 = \rho_{P_1}\,(\eta_{l^*,L_1}\,(l^*(x)\oplus t_1)).$$

This operation constructs a new union node with the two children being $x$ and the clique-width expression for $\mathcal{T}_1$. The $\eta_{l^*,L_1}$ operation adds all appropriate edges to $x$, and the $\rho_{P_1}$ operation "collapses" label class $X\cup\{x\}$ into $X$ for each $X\subseteq\{s_2,s_3,\dots,s_k\}$.

**Step 2.** For $i$ from 2 to $k$ consider the subtrees $\mathcal{T}_i$ rooted at the $-s_i$ children of $x$. We let $t'_{i-1}$ denote the clique-width expression formed after considering $\mathcal{T}_1 \cup \cdots \cup \mathcal{T}_{i-1}$. For each $i$, let $L_i$ be the set of labels $\{l_{X\cup\{x\}} : X \subseteq \{s_1, s_2, \ldots, s_{i-1}, s_{i+1}, \ldots, s_k\}\}$ and let $P_i$ be the set of pairs of labels $\{(l_{X\cup\{x\}}, l_X) : X \subseteq \{s_1, s_2, \ldots, s_{i-1}, s_{i+1}, \ldots, s_k\}\}$. The following operation will form $t'_i$ from the clique-width expression $t'_{i-1}$ by incorporating $t_i$, the clique-width expression for $\mathcal{T}_i$ (recall that $l^*$ is the label of $x$):

$$t'_i = \rho_{P_i}\ (\eta_{l^*,L_i}\ (t'_{i-1} \oplus t_i)).$$

This operation constructs a new union node with the two children being $t'_{i-1}$ and the clique-width expression for $\mathcal{T}_i$. As in Step 1, via a series of $\eta$ operations we add all the edges from appropriate vertices in $\mathcal{T}_i$ to vertex $x$ and then "collapse" label class $X \cup \{x\}$ into $X$ for each $X \subseteq \{s_1, s_2, \ldots, s_{i-1}, s_{i+1}, \ldots, s_k\}$.

After the loop ends, we set $t_x = t'_k$.

**Step 3.** Suppose that the children of the root $R$ are $\{x_1, x_2, \ldots, x_j\}$ and that each of $T_{x_1}, T_{x_2}, \ldots, T_{x_j}$ has been processed as above, resulting in clique-width expressions $t_{x_1}, t_{x_2}, \ldots, t_{x_j}$, respectively. Each vertex in $T_{x_1} \cup T_{x_2} \cup \cdots \cup T_{x_j}$ is adjacent to some subset of $\{1, 2, \ldots, k\}$, where all vertices adjacent to a particular subset have the same label. Let $l^*_1, l^*_2, \ldots, l^*_k$ denote new labels (they will be used in the creation of the vertices $1, 2, \ldots, k$) and define set $L_i$, $1 \le i \le k$, to be the union of $\{l^*_h : i < h \le k\}$ and the labels corresponding to subsets of $\{1, \ldots, k\}$ that contain vertex $i$. (Note that $L_i$ represents the vertices that are adjacent to $i$.)

The clique-width expression for the entire graph is

$$t_R = \eta_{l^*_k,L_k}\ (\cdots \eta_{l^*_2,L_2}\ (\eta_{l^*_1,L_1}\ (l^*_1(1) \oplus \cdots \oplus l^*_k(k) \oplus t_{x_1} \oplus \cdots \oplus t_{x_j}))\cdots).$$

Note that this operation forms the vertices in the clique $K$ with the $l^*_i(i)$ operations and forms the union of all these vertices and the $\{t_{x_i}\}$ expressions. Finally, the $\eta_{l^*_i,L_i}$ metaoperations add the edges in $K$ as well as the edges between $K$ and the rest of the graph.

*Example* 3.1. Consider the graph in Figure 4 and its construction tree presented in Figure 5. The $3*2^{2-1} = 6$ labels used by the algorithm will be denoted $a, b, c, d, e, f$. We construct the clique-width expression for the graph as indicated by the algorithm from bottom to top. At each step we indicate the set of vertices corresponding to the labels $a, b, \ldots, f$. Note that the same label (e.g., $a$) can correspond to different subsets of vertices at different stages of the algorithm.

When the algorithm processes $T_{10}$ (i.e., the leaf 10) it creates vertex number 10 and gives it label $a$. That is, the clique-width expression corresponding to $T_{10}$ is $a(10)$, where label $a$ corresponds to the set $\{5, 7\}$ consisting of 5 (the source of 10) and 7 (the parent of 10). When the algorithm processes $T_7$ (in Step 1) the clique-width expression for $T_7$ will be $t'_1 = \rho_{P_1}\ (\eta_{l^*,L_1}\ (l^*(7) \oplus t_1))$. In the last formula $t_1$ is the clique-width expression for $T_{10}$, which is $a(10)$, and the label $l^*$ is a new label corresponding to the set $\{2, 5\}$ that we denote by $b$. The sequence $\eta_{l^*,L_1}$ will reduce to $\eta_{b,a}$, and the sequence $\rho_{P_1}$ will reduce to renaming the label corresponding to the set $\{5, 7\}$ to the label corresponding to the set $\{5\}$. Since we do not have (in the current expression) a label corresponding to the set $\{5\}$ we can omit the $\rho_{P_1}$ from the formula and remember that label $a$ now corresponds to the set $\{5\}$. Thus the clique-width expression for $T_7$ is $\eta_{a,b}\ (b(7) \oplus a(10))$, where label $a$ corresponds to the set $\{5\}$ and label $b$ corresponds to the set $\{2, 5\}$.

Similarly, the clique-width expression obtained by the algorithm for $T_5$ is $\eta_{a,c}\ (\eta_{b,c}\ (c(5) \oplus \eta_{a,b}\ (b(7) \oplus a(10))))$, where label $c$ corresponds to $\{2, 3\}$, label $a$ corresponds to $\{\}$, and label $b$ corresponds to $\{2\}$. We denote the last expression by $z_5$.

When the algorithm processes $T_6$, the clique-width expression obtained is $\eta_{a,b}\ (b(6) \oplus a(8))$, where $a$ corresponds to $\{3\}$ and $b$ corresponds to $\{2, 3\}$. We denote the last expression by $z_6$. Since $T_6$ and $T_5$ have the same parent, the vertices in $val(z_6)$ are renamed in order to follow the assumption of the algorithm that labels corresponding to the same subset of vertices in $val(z_5)$ and in $val(z_6)$ are unique. Since label $b$ of $val(z_6)$ is the same as label $c$ of $val(z_5)$, label $b$ of $val(z_6)$ is renamed to $c$. Since label $a$ of $val(z_6)$ is different from label $a$ of $val(z_5)$, label $a$ of $val(z_6)$ is renamed to $d$. Thus the algorithm sets $z_6 = \rho_{b \to c}\ (\rho_{a \to d}\ (z_6))$, where labels $c$ and $d$ of $val(z_6)$ correspond to the sets $\{2, 3\}$ and $\{3\}$, respectively.

We now consider the actions of Steps 1 and 2 on $T_3$. The source of 3 is 1 and $p(3) = 2$. By definition, $\mathcal{T}_1 = T_5 \cup T_6$ and $\mathcal{T}_2 = T_9$, and $t_1$ and $t_2$ are the clique-width expressions of $\mathcal{T}_1$ and $\mathcal{T}_2$, respectively. The clique-width expression $t_1$ is the union of the clique-width expressions corresponding to $T_5$ and $T_6$. Thus $t_1 = z_5 \oplus z_6$, where labels $a$, $b$, $c$, and $d$ of $val(t_1)$ correspond to the sets $\{\}$, $\{2\}$, $\{2, 3\}$, and $\{3\}$, respectively. Similarly, $t_2 = e(9)$, where label $e$ corresponds to the set $\{1, 3\}$.

Now in Step 1, vertex 3 is processed together with $t_1$, yielding $t'_1 = \rho_{c \to b}\ (\rho_{d \to a}\ (\eta_{f,d}\ (\eta_{f,c}\ (f(3) \oplus t_1))))$, where labels $a$, $b$, and $f$ correspond to sets $\{\}$, $\{2\}$, and $\{1, 2\}$, respectively.

Step 2 yields $t_3 = t'_2 = \eta_{f,e}\ (t'_1 \oplus e(9))$, where labels $a$, $b$, $e$, and $f$ correspond to the sets $\{\}$, $\{2\}$, $\{1\}$, and $\{1, 2\}$, respectively.

Finally, we examine the outcome of Step 3, which calculates the clique-width expression of the graph. Note that $t_4 = f(4)$, where the label $f$ corresponds to the set $\{1, 2\}$:

$$t_R = \eta_{b,c}\ (\eta_{e,d}\ (\eta_{f,c}\ (\eta_{f,d}\ (\eta_{c,d}\ (c(2) \oplus d(1) \oplus t_3 \oplus t_4))))).$$

*Proof of correctness.* We now show that the above algorithm will construct an expression tree for $G$ and will use no more than $3 * 2^{k-1}$ labels. The fact that $G$ is the value of the clique-width expression constructed by the algorithm follows by a straightforward induction argument.

LEMMA 3.2. *The algorithm requires at most* $3 * 2^{k-1}$ *labels.*

*Proof.* First we note that a clique-width expression $t$ uses at most $h$ labels if and only if every subexpression $t'$ of $t$ uses at most $h$ labels. So, we assume by induction that for a nonroot vertex $x \in T_G$, all subexpressions $t_1, t_2, \ldots, t_k$ use at most $3 * 2^{k-1}$ labels and show that this is also true for the clique-width expression built for $t_x$ by the above algorithm. We also assume by induction that for $1 \le i \le k$, $val(t_i)$ has at most $2^k$ labels, and that these labels correspond to subsets of $\{s_1, \ldots, s_{i-1}, s_i, \ldots, s_k, x\}$. Clearly these assumptions hold for the leaves of $T_G$.

We will show that $val(t_x)$ has at most $2^k$ labels and that these labels correspond to subsets of $\{s_1, \ldots, s_k\}$. We will also show that $t_x$ does not use more than $2^k + 2^{k-1}$ $(= 3 * 2^{k-1})$ labels.

First by induction we assume that $t_1$ does not use more than $3 * 2^{k-1}$ labels and that the $val(t_1)$ labels correspond to the subsets of $\{s_2, \ldots, s_k, x\}$. Thus $val(l^*(x) \oplus t_1)$ has at most $2^k + 1$ labels. Due to the $\rho_{P_1}$ operation in Step 1 of the algorithm, all the labels of $val(t'_1)$ correspond to some (but not all) subsets of $\{s_1, \dots, s_k\}$. Now examine the clique-width expression $t_i$, $i > 1$ (Step 2). By induction, all $val(t_i)$ labels correspond to subsets of $\{s_1, \dots, s_{i-1}, s_{i+1}, \dots, s_k, x\}$. All $val(t'_{i-1})$ labels correspond to subsets of $\{s_1, \dots, s_k\}$. Thus the only new labels that $val(t'_{i-1})$ can add (to the labels of $val(t_i)$) are those labels that contain $s_i$ and there are at most $2^{k-1}$ such labels. Thus the graph $val(t'_{i-1} \oplus t_i)$ has at most $2^k + 2^{k-1}$ labels. Again, due to the $\rho_{P_i}$ operation, all the labels of $val(t'_i)$ are subsets of $\{s_1, \dots, s_k\}$. Thus the subexpression $t'_k$ $(= t_x)$ uses at most $3 * 2^{k-1}$ labels and $val(t_x)$ has at most $2^k$ labels, corresponding to the subsets of $\{s_1, \dots, s_k\}$, as required.

We now turn our attention to the root (Step 3 of the algorithm). In particular we show that the algorithm uses no more than $3 * 2^{k-1}$ labels. The fact that the algorithm requires no more than $3 * 2^{k-1}$ labels to process the children of the root follows immediately from the argument above. Note that equality may be achieved when $i = k$. To process the root, we note that $2^k$ labels (i.e., corresponding to all subsets of $\{1, 2, \dots, k\}$) are needed in $t_{x_1} \cup t_{x_2} \cup \cdots \cup t_{x_j}$. Similarly, $k$ different labels (i.e., $l^*_1, l^*_2, \dots, l^*_k$) are needed to label $1, 2, \dots, k$. Since $2^k + k \leq 3 * 2^{k-1}$ for $k \geq 1$, the result follows. $\square$

To finish the proof of Theorem 1.2 we just have to show that the algorithm will also work for partial $k$-trees. To see this, we first note that a partial $k$-tree may be constructed in a similar way as a $k$-tree. In particular, given a partial $k$-tree $G$, consider the construction tree $T_{G'}$ for $G'$, a $k$-tree that contains $G$. Since any initial graph on $k$ vertices, or fewer, can be formed using at most $k$ labels and the algorithm does not depend on the subset of existing vertices to which a vertex is adjacent being a clique, we see that a slight modification to the algorithm (namely, allowing each vertex to be adjacent to at most $k$, rather than exactly $k$, ancestors) solves the problem for partial $k$-trees as well.

## 4. Lower bound.

The purpose of this section is to establish the lower bound stated in Theorem 1.3.

THEOREM 1.3. *For any $k$, there is a graph $G$ where $twd(G) = k$ and $cwd(G) \geq 2^{\lfloor k/2 \rfloor - 1}$.*

Since the clique-width of any graph is at least one, the theorem is clearly true for $k$ at most three. Our proof assumes $k$ is at least four, and in particular, we define a $k$-tree $F$ and prove that $cwd(F) \geq 2^{\lfloor k/2 \rfloor - 1}$.

Using the notation of section 2, we define the $k$-tree $F$ by its construction tree $T_F$. Let the vertices of $F$ be numbered $\{1, \dots, n\}$. Like any $k$-tree, $F$ can be constructed by starting with the initial $k$-clique $\{1, \dots, k\}$ and at each step adding a new vertex $i$ that is universal to a $k$-clique in the intermediate graph, which we denote by $Q_i$.

The tree $T_F$ corresponds to the construction mentioned above. In the following we denote $T_F$ by $T$. Each node $i$ of the tree $T$ corresponds to vertex $i$ of $F$. The root of $T$ is denoted by $R$ and corresponds to vertex $k$ of $F$. The root $R$ has one child denoted by $B$ that corresponds to vertex $k+1$ of $F$. Clearly, $Q_B = \{1, 2, \dots, k\}$.

Let $L^j$ denote the set of all nodes of the tree $T$ that are at distance $j$ from the root $R$. We call $L^j$ the $j$th level of the tree $T$. Clearly $L^0 = \{R\}$ and $L^1 = \{B\}$. The tree $T$ has $\alpha$ levels, where $\alpha = (2k+1)2^{\lfloor k/2 \rfloor + 1}$. For $2 \leq l \leq \alpha$, the $l$th level of the tree $T$, $L^l$ is defined as follows:

- For every node $x$ of the $(l-1)$st level $L^{l-1}$, add $k$ new nodes $\{x_1, x_2, \dots, x_k\}$ to the $l$th level $L^l$, all of which are children of $x$ such that the following holds:
  - Let $Q_x = \{y_1, \dots, y_k\}$ be the $k$-clique to which $x$ is universal. Then for $1 \leq i \leq k$, $x_i$ is the $-y_i$ child of $x$. In other words, $x_i$ is universal to the $k$-clique $Q_{x_i} = Q_x \cup \{x\} - \{y_i\}$.

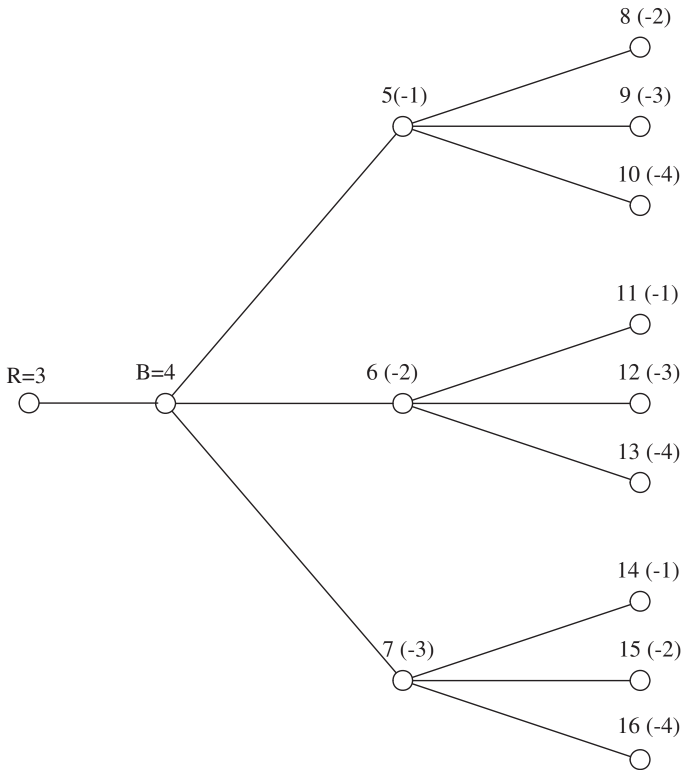


FIG. 6. *The beginning of the construction tree $T$ for the graph $F$ when $k$ is three.*

Note that all the leaves of $T$ are in level $L^\alpha$ and all internal nodes of $T$, except the root $R$, have exactly $k$ children.

*Example* 4.1. Figure 6 illustrates the beginning of the construction tree $T$ for the graph $F$ when $k$ is three. The root of this tree is $R$ which corresponds to vertex 3 of $F$. The root $R$ has one child $B$ which corresponds to vertex 4 of $F$. Level $L^0 = \{R\}$, level $L^1 = \{B\}$, level $L^2 = \{5, 6, 7\}$, and level $L^3 = \{8, 9, \dots, 16\}$. We omit from the figure the $k$-clique to which each node of the tree is universal. Instead, we employ a $(-i)$ label for a node $u$ indicating that $u$ is the $-i$ child of its parent. For example, node 12 is the $-3$ child of node 6, which is the $-2$ child of node 4. The $k$-clique to which node 12 is universal is $Q_{12} = \{1, 4, 6\}$. The parent of vertex 12 is $p(12) = 6$. Figure 7 illustrates the graph corresponding to the construction tree $T$ of Figure 6. Note that the graph of Figure 7 is the subgraph of the graph $F$ for $k$ equals three, induced by the vertices $\{1, 2, \dots, 16\}$.

The following four facts follow from the above definitions.

FACT 4.2. *Let $u, v$ be any two vertices of $F$ not in $Q_B = \{1, \dots, k\}$, such that $u$ is adjacent to $v$ in $F$. Then $u$ is either an ancestor or a descendant of $v$ in $T$.*

*Proof.* In the construction of $F$, whenever a new vertex (say $x$) is added it is made universal to a $k$-clique of vertices (denoted $Q_x$) that were already added to $F$. If a vertex $y$ was added to $F$ before $x$, then $x$ is adjacent to $y$ if and only if $y$ is in $Q_x$. An easy induction on the construction of the tree $T$ shows that for every node $x$ of $T$ that is not in $Q_B$, all the vertices of $Q_x$ that are not in $Q_B$ are ancestors of $x$ in $T$. Let $u$ and $v$ be two nodes of $T$ that are not in $Q_B$, such that vertices $u$ and $v$ are

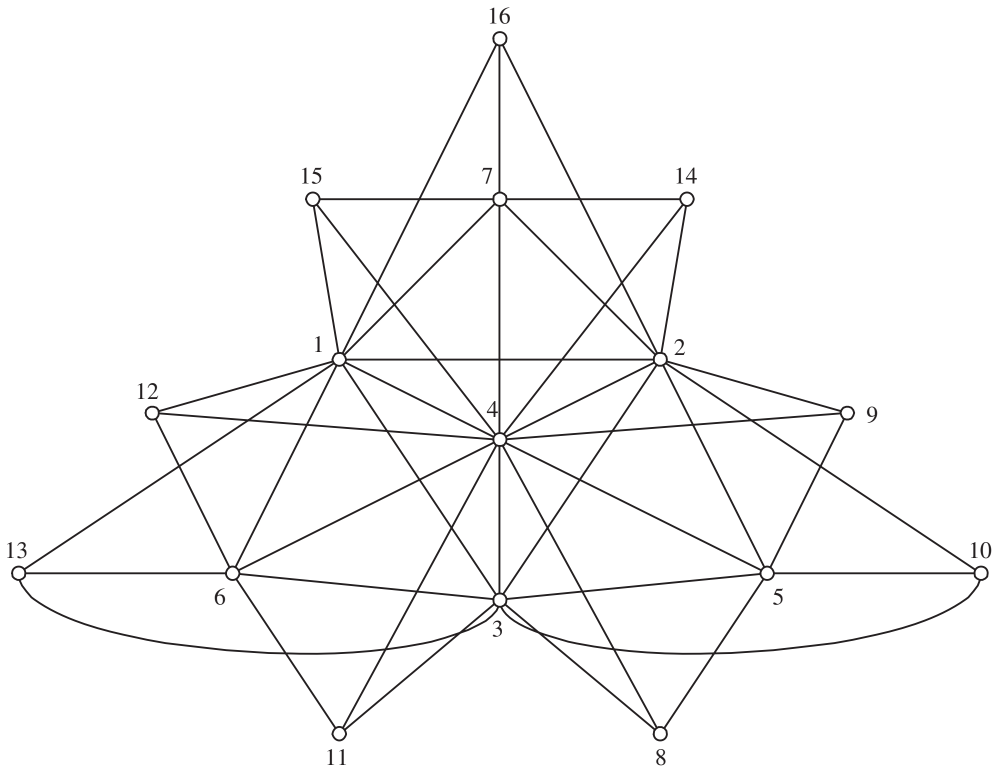


FIG. 7. *The graph corresponding to the construction tree of Figure* 6.

adjacent in $F$. Assume without loss of generality that $u$ was added to $F$ before $v$. Since $u$ is adjacent to $v$, $u$ must be in $Q_v$. By the above argument, all vertices of $Q_v$ are ancestors of $v$. Thus, $u$ must be an ancestor of $v$. □

FACT 4.3. *Let $u, w$ be any two adjacent vertices of $F$ not in $Q_B = \{1, \ldots, k\}$, such that $w$ is a descendant of $u$ in $T$. Then $u$ is in $Q_w$.*

*Proof.* From the construction of $F$, vertex $u$ is added to $F$ before $w$. When $w$ is added to $F$, it is made adjacent to all vertices of the $k$-clique $Q_w$. All the vertices of $F$ that are not in $Q_w$ and were added to $F$ before $w$ cannot be adjacent to $w$. Since $u$ is adjacent to $w$ we conclude that $u$ must be in $Q_w$. □

FACT 4.4. *Let $u, v, w$ be any three different vertices of $F$ not in $Q_B = \{1, \ldots, k\}$, such that $v$ is a descendant of $u$ in $T$, $w$ is a descendant of $v$ in $T$, $v$ is not adjacent to $u$ in $F$, and $w$ is adjacent to $v$ in $F$. Then $w$ is not adjacent to $u$ in $F$.*

*Proof.* Suppose $w$ is adjacent to $u$ in $F$. By Fact 4.3, both $v$ and $u$ belong to $Q_w$. Thus since $Q_w$ is a clique, $u$ must be adjacent to $v$ in $F$, a contradiction. □

FACT 4.5. *Let $u, w$ be any two vertices of $F$ not in $Q_B = \{1, \ldots, k\}$, such that $w$ is a child of $u$ in $T$. Then $Q_u - \{p(u)\} = Q_w - \{p(w)\}$ if and only if $w$ is the $-p(u)$ child of $u$.*

*Proof.* Suppose $w$ is the $-p(u)$ child of $u$. By definition, $Q_w = Q_u \cup \{u\} - \{p(u)\}$. Since $u = p(w)$ we conclude that $Q_w - \{p(w)\} = Q_u - \{p(u)\}$.

Suppose $Q_u - \{p(u)\} = Q_w - \{p(w)\}$. If $w$ is not the $-p(u)$ child of $u$, then $Q_w = Q_u \cup \{u\} - \{t\}$ for some vertex $t \in Q_u$ different from $p(u)$. Substituting $u = p(w)$, we see that $Q_w - \{p(w)\} = Q_u - \{t\}$, which contradicts $Q_u - \{p(u)\} = Q_w - \{p(w)\}$.

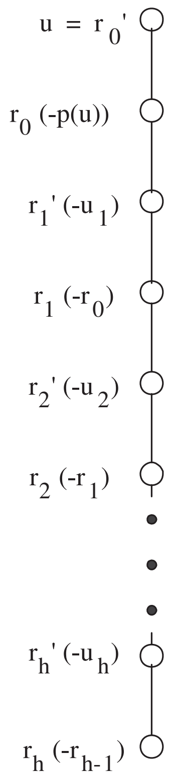


FIG. 8. *The nodes of the tree $T_u$ defined in the proof of Claim* 4.6.

Thus, $w$ must be the $-p(u)$ child of $u$. ◻

For a node $x$ of $T$, we denote by $T_x$ the subtree of $T$ rooted at $x$. We denote by $L_x^j$ the $j$th level of the tree $T_x$. We say that $T_x$ is a *complete tree* if $L_x^{2k+1}$ is not empty (i.e., the distance in $T$ between $x$ and the root $R$ is at most $\alpha - (2k+1)$). We now establish some properties of complete trees that will be used in subsequent arguments.

CLAIM 4.6. *Let $u$ be a node of $T$ other than the root $R$, such that $T_u$ is a complete tree in $T$. Then for every $S \subseteq Q_u - \{p(u)\}$, there exists vertex $r_S \in N_F[u] \cap T_u$ such that $(Q_{r_S} - \{p(r_S)\}) \cap (Q_u - \{p(u)\}) = S$. Furthermore, these $r_S$ vertices can be chosen such that the set $\mathcal{S} = \{r_S : S \subseteq Q_u - \{p(u)\}\} - \{u\}$ is an independent set.*

*Proof.* If $S = Q_u - \{p(u)\}$, then we take $r_S = u$. Now let $\{u_1, u_2, \ldots, u_h\} = Q_u - (\{p(u)\} \cup S)$. We define nodes in the subtree $T_u$ as presented in Figure 8. Let $r_0$ be the $-p(u)$ child of $u$. By Fact 4.5, $Q_{r_0} - \{u\} = Q_u - \{p(u)\}$. For $i$ from 1 to $h$, we let $r_i'$ be the $-u_i$ child of $r_{i-1}$ and we let $r_i$ be the $-r_{i-1}$ child of $r_i'$. Since $r_{i-1} = p(r_i')$, by Fact 4.5, $Q_{r_i} - \{p(r_i)\} = Q_{r_i'} - \{p(r_i')\}$. We set $r_S = r_h$ and we show that $r_S$ satisfies the required properties. Let $r_0' = u$. Now, by an easy induction on $i$, where $1 \le i \le h$,

$$Q_{r_i} - \{p(r_i)\} = Q_u \cup \{r_j' : 0 \le j < i\} - (\{p(u)\} \cup \{u_1, \ldots, u_i\}).$$

Thus, setting $r_S = r_h$ we see that $r_S$ satisfies the required properties.

We now show that $\mathcal{S}$ is an independent set. Suppose there exist two adjacent vertices $r_{S_1}$ and $r_{S_2}$ in $\mathcal{S}$. By Fact 4.2, $r_{S_1}$ is either an ancestor or a descendant

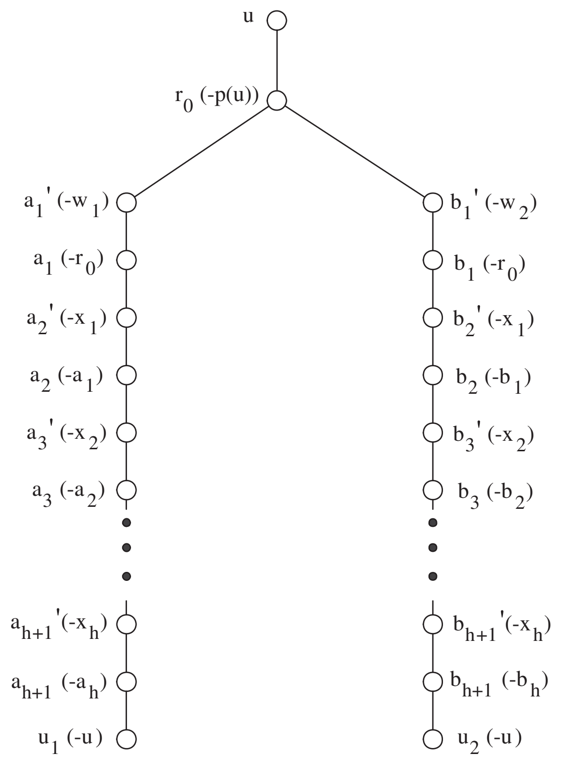


FIG. 9. *The nodes of the tree $T_u$ defined in the proof of Claim* 4.7.

of $r_{S_2}$ in $T$. Assume without loss of generality that $r_{S_1}$ is an ancestor of $r_{S_2}$ (i.e., $S_2 \subset S_1$). By Fact 4.3, $r_{S_1} \in Q_{r_{S_2}}$. By the above construction, there exist integers $x$ and $y$ such that $1 \leq y < x$, $Q_u - (\{p(u)\} \cup S_2) = \{u_1, u_2, \ldots, u_y, \ldots, u_x\}$, $r_{S_2} = r_x$, $Q_u - (\{p(u)\} \cup S_1) = \{u_1, u_2, \ldots, u_y\}$, and $r_{S_1} = r_y$. From the above formula

$$Q_{r_{S_2}} - \{p(r_{S_2})\} = Q_u \cup \{u, r'_1, \ldots, r'_{x-1}\} - (\{p(u)\} \cup \{u_1, \ldots, u_x\}),$$

which implies that $r_{S_1} \notin Q_{r_{S_2}}$, a contradiction. ▯

CLAIM 4.7. *Let $u$ be a node of $T$ other than the root $R$, such that $T_u$ is a complete tree. Then for any $W \subset Q_u - \{p(u)\}$ with $w_1, w_2 \in W$, there exist vertices $u_1, u_2 \in T_u$, at level at most $2k+1$ of $T_u$, such that $u_1$ is neither an ancestor nor a descendant of $u_2$, $Q_{u_i} - \{p(u_i)\} = (W - \{w_i\}) \cup Y_{u_i}$, where $i = 1, 2$, $Y_{u_i} \cap W = \emptyset$, all vertices of $Y_{u_i}$ are in $T_u$, $Y_{u_1} \cap Y_{u_2} = \emptyset$, and no vertex of $Y_{u_1}$ is an ancestor or a descendant of a vertex in $Y_{u_2}$.*

*Proof.* Let $Q_u - \{p(u)\} - W = \{x_1, \ldots, x_h\}$. Thus,

(1) $$W - \{w_1\} = Q_u - \{p(u), w_1, x_1, \ldots, x_h\},$$

(2) $$W - \{w_2\} = Q_u - \{p(u), w_2, x_1, \ldots, x_h\}.$$

We now define nodes in the subtree $T_u$ as presented in Figure 9. Let $r_0$ be the $-p(u)$ child of $u$. Let $a'_1$ be the $-w_1$ child of $r_0$ and let $a_1$ be the $-r_0$ child of $a'_1$. Let $b'_1$ be the $-w_2$ child of $r_0$ and let $b_1$ be the $-r_0$ child of $b'_1$. For $i$ from 2 to $h+1$, let $a'_i$ be the $-x_{i-1}$ child of $a_{i-1}$, let $b'_i$ be the $-x_{i-1}$ child of $b_{i-1}$, let $a_i$ be the $-a_{i-1}$ child of $a'_i$, and let $b_i$ be the $-b_{i-1}$ child of $b'_i$. We set $u_1$ as the $-u$ child of $a_{h+1}$ and $u_2$ as the $-u$ child of $b_{h+1}$. As in the proof of Claim 4.6 we get the following formulas:

$$Q_{a_{h+1}} - \{p(a_{h+1})\} = Q_u \cup \{u, a'_1, \dots, a'_h\} - (\{p(u)\} \cup \{w_1, x_1, \dots, x_h\}), \tag{3}$$

$$Q_{b_{h+1}} - \{p(b_{h+1})\} = Q_u \cup \{u, b'_1, \dots, b'_h\} - (\{p(u)\} \cup \{w_2, x_1, \dots, x_h\}). \tag{4}$$

Since $Q_{u_1} = Q_{a_{h+1}} \cup \{a_{h+1}\} - \{u\}$ we obtain from formula (3) that

$$Q_{u_1} - \{p(u_1)\} = Q_u \cup \{a'_1, \dots, a'_{h+1}\} - (\{p(u)\} \cup \{w_1, x_1, \dots, x_h\}). \tag{5}$$

From formulas (1) and (5) we get

$$Q_{u_1} - \{p(u_1)\} = (W - \{w_1\}) \cup \{a'_1, \dots, a'_{h+1}\}. \tag{6}$$

Similarly, from formulas (2) and (4),

$$Q_{u_2} - \{p(u_2)\} = (W - \{w_2\}) \cup \{b'_1, \dots, b'_{h+1}\}. \tag{7}$$

Thus, setting $Y_{u_1} = \{a'_1, \dots, a'_{h+1}\}$ and $Y_{u_2} = \{b'_1, \dots, b'_{h+1}\}$ we get from formulas (6) and (7) that $Q_{u_i} - \{p(u_i)\} = (W - \{w_i\}) \cup Y_{u_i}$, $i = 1, 2$. Since all the vertices of $Y_{u_i}$, $i = 1, 2$, are below $u$ and all vertices of $W$ are above $u$ in the tree $T$, $W \cap Y_{u_i} = \emptyset$, $i = 1, 2$. Clearly, $Y_{u_1} \cap Y_{u_2} = \emptyset$ and no vertex of $Y_{u_1}$ is an ancestor or a descendant of a vertex in $Y_{u_2}$. $\square$

For a node $x$ of $T$ other than the root $R$, we let $P_x$ denote a path $(x = x_0, x_1, \dots, x_\beta)$ of $T_x$, where $x_\beta$ is a leaf and

($\star$)
- $Q_{x_i} - \{p(x_i)\} = Q_{x_{i-1}} - \{p(x_{i-1})\}$, $i > 0$,
- $p(x_i) = x_{i-1}$, $i > 0$.

FACT 4.8. *Let $x$ be a node of $T$ other than the root $R$ and let $u$ and $v$ be two nonadjacent nodes on the path $P_x$ (that satisfies ($\star$)). Then $u$ and $v$ are not adjacent in $F$.*

*Proof.* We prove the claim by induction on the distance between $u$ and $v$ on the path $P_x$. Since $u$ and $v$ are not adjacent on the path, their distance on the path must be at least two. Assume without loss of generality that $v$ is a descendant of $u$.

Suppose the distance between $u$ and $v$ on the path $P_x$ is two. Since $P_x$ satisfies ($\star$), $v$ is the $-u$ child of $p(v)$, which implies that $u$ is not in $Q_v$. By Fact 4.3, $v$ cannot be adjacent to $u$ in $F$.

Suppose the distance between $u$ and $v$ on the path $P_x$ is greater than two. We can assume, by the induction hypothesis, that $p(v)$ is not adjacent to $u$ in $F$. Now since $v$ is adjacent to $p(v)$ in $F$ and $p(v)$ is not adjacent to $u$ in $F$ we get by Fact 4.4 that $v$ is not adjacent to $u$ in $F$. $\square$

For any node $x$ of $T$ other than the root $R$, where $T_x$ is a complete tree and $Q \subseteq Q_x - \{p(x)\}$, we let $X_x^Q = \{y : y \in L_x^{2k+1} \wedge (Q_y - \{p(y)\}) \cap (Q_x - \{p(x)\}) = Q\}$ (i.e., $X_x^Q$ denotes the set of vertices at level $2k + 1$ whose sources contain $Q$ and no vertices in $Q_x - \{p(x)\} - Q$). Furthermore, we let $P_x^Q = \{P_y : y \in X_x^Q\}$ (i.e., $P_x^Q$ denotes the paths rooted at $y \in X_x^Q$ that satisfy ($\star$) above).

CLAIM 4.9. *Let $x$ be a node in $T$ other than the root $R$, such that $T_x$ is a complete tree, and let $Q \subseteq Q_x - \{p(x)\}$. Then $|X_x^Q| \geq (k - |Q|)^{k+1}$.*

*Proof.* Let $Y^j$, $1 \leq j \leq k + 1$, denote the set of all vertices at level $j$ (of $T_x$) that include all the vertices of $Q$ in their sources. In other words, $Y^j = \{y : y \in L_x^j \wedge Q \subseteq Q_y - \{p(y)\}\}$. Note that any vertex $z$ at level $j$, where $1 \leq j < k + 1$, is created by making it adjacent to $k$ vertices, $|Q|$ of which are forced. Thus $z$ has $k-|Q|$ children that are in $Y^{j+1}$. It follows by a straightforward induction on $j$ that $|Y^j| = (k-|Q|)^j$, $1 \le j \le k+1$.

Let $y$ be a vertex of $Y^{k+1}$. It is easy to see that we can construct a path of length $k$ from $y$ to a vertex $y'$ of level $2k+1$ of $T_x$ such that $Q_{y'} - \{p(y')\} \cap Q_x - \{p(x)\} = Q$. Thus, each vertex $y$ of $Y^{k+1}$ corresponds to a unique vertex $y'$ of $X_x^Q$, implying that $|X_x^Q| \ge |Y^{k+1}| = (k-|Q|)^{k+1}$. $\square$

In order to argue about the clique-width of $F$, we let $\mathcal{T}$ denote an optimum clique-width parse tree for $F$. For $a$ an internal node of $\mathcal{T}$, we let $\mathcal{T}_a$ denote the subtree of $\mathcal{T}$ rooted at $a$ and let $V_a$ represent the leaves (i.e., vertices of $F$) of $\mathcal{T}_a$. As the following claim shows, having a set of vertices outside $V_a$ together with certain neighbors inside $V_a$ establishes a lower bound on the number of distinct labels required at $a$ (i.e., to label the vertices of $V_a$ at node $a$ of $\mathcal{T}$).

CLAIM 4.10. *Let $X$ be a set of $l$ vertices in $V - V_a$, where for each subset $Y$ of these vertices, there is a vertex inside $V_a$ adjacent to all vertices of $Y$ and to no other vertices in $X$. Then at least $2^l$ labels are required to label the vertices of $V_a$.*

*Proof.* For any two vertices $x, y$ inside $V_a$ adjacent to different subsets of $X$, there is at least one vertex outside $V_a$ that is adjacent to $x$ and not to $y$. Thus $x$ and $y$ must have different labels at position $a$ of the parse tree. Since there are $2^l$ such vertices, the result follows. $\square$

For $\mathcal{T}_1$ a subtree of $\mathcal{T}$ and $P_y$ a path of $T$ satisfying ($\star$), we say that $P_y$ is *full with respect to* $\mathcal{T}_1$ if all vertices of $P_y$ are in $\mathcal{T}_1$. Given a set $P_x^Q$, we will often choose a lowest possible internal node of $\mathcal{T}$, say $a$, such that some path in $P_x^Q$ is full with respect to $\mathcal{T}_a$ (i.e., for some path $P \in P_x^Q$ all of its vertices are in $V_a$; $P$ is said to be *full with respect to* $a$). If $T_u \cap V_a = \emptyset$ for some $u$, we say that $T_u$ is *empty with respect to* $a$.

CLAIM 4.11. *Assume $cwd(F) < 2^{\lfloor k/2 \rfloor -1}$. Let $x$ be a node in $T$ other than the root $R$, where $T_x$ is a complete tree, let $Q$ be a strict subset of $Q_x - \{p(x)\}$, and let $a$ be a lowest node in $\mathcal{T}$ such that some path in $P_x^Q$ is full with respect to $a$. Then there is a vertex $y \in X_x^Q$ such that $T_y$ is empty with respect to $a$.*

*Proof.* From the definition of $a$ it follows that $a$ must be an $\oplus$ node. Let $l$ and $r$ be the left and the right child of $a$ in the tree $\mathcal{T}$, respectively. From the definition of $a$ it follows that for each path $P$ in $P_x^Q$ that is full with respect to $a$, the vertices of $P$ are split between the two subtrees $\mathcal{T}_l$ and $\mathcal{T}_r$. Thus, for each path in $P_x^Q$ that is a full path with respect to $a$ in $\mathcal{T}$, at least one edge of the path (in $F$) must be created by an operation above $a$. Thus each such full path requires at least two new labels and these labels cannot be shared by other full paths with respect to $a$ in $P_x^Q$. Since $cwd(F) < 2^{\lfloor k/2 \rfloor -1}$, there are fewer than $2^{\lfloor k/2 \rfloor -2}$ full paths with respect to $a$ in $P_x^Q$. If a tree rooted at $z \in X_x^Q$ is neither empty nor has a full path with respect to $a$, then there is some vertex $w \in T_z$ that is in $V_a$, but $w'$, a neighbor of $w$ (in $T_z$), is not in $V_a$. Thus $w$ must have a new label and its label must be different from all of the pairs of labels required for the full paths and all of the labels required for other "partially present" trees. Now, the number of such partially present trees is less than $2^{\lfloor k/2 \rfloor}$.

Thus in $X_x^Q$ fewer than $2^{\lfloor k/2 \rfloor -2}$ vertices are the roots of full paths and fewer than $2^{\lfloor k/2 \rfloor}$ are the roots of partially present trees. By Claim 4.9, $|X_x^Q| \ge 2^{k+1}$ and thus there is at least one vertex that is the root of an empty tree with respect to $a$. $\square$

In order to prove Theorem 1.3 we will examine the $k$-tree $F$ and proceed by assuming $cwd(F) < 2^{\lfloor k/2 \rfloor -1}$ until we finally reach a contradiction.

Initially we will examine the vertices in $X_B^{2,\dots,k-1}$ and let $a$ be a lowest node in $\mathcal{T}$

such that some path in $P_B^{2,\ldots,k-1}$ is full with respect to $a$. We will then show that at least $\lfloor k/2 \rfloor$ vertices of $\{2,\ldots,k-1\}$ are in $V_a$ and have unique labels (unique in the sense that no two such vertices can have the same label) at $a$. We then proceed by induction by identifying other vertices in $T$ and looking at $b$, a lowest node in $\mathcal{T}$ such that some path rooted at one of these vertices is full with respect to $b$. We show that $V_a \subseteq V_b$ and that the set of vertices requiring unique labels at $a$ also requires unique labels at $b$ and that this set must be augmented by a new vertex that requires a new label. This augmentation continues until eventually this set has cardinality equal to $2^{\lfloor k/2 \rfloor -1}$, contradicting the assumption that $cwd(F) < 2^{\lfloor k/2 \rfloor -1}$, and thereby proving the theorem. The depth of $T$, $\alpha$ is chosen so that all subtrees used to generate full paths are complete and have depth of at least $2^{\lfloor k/2 \rfloor} + 2k + 1$. The length of the full paths generated by these subtrees will be at least $2^{\lfloor k/2 \rfloor}$, since the root of each such path is at level $2k+1$ of its corresponding subtree.

Following the above outline, we start by establishing some claims about $\mathcal{T}_a$.

CLAIM 4.12. *At least $\lfloor k/2 \rfloor$ vertices of $Q_B - \{1,k\}$ (i.e., $\{2,\ldots,k-1\}$) are in $V_a$ and each must have a unique label at $a$.*

*Proof.* Suppose there is a full path $P$ with respect to $a$ rooted at $z \in X_B^{Q_B-\{1,k\}}$. Since such a path has more than $2^{\lfloor k/2 \rfloor}$ vertices, at least two vertices $x$ and $y$ on this path must have the same label at $a$. Suppose $x$ is closer to $z$ than is $y$. Since $z \in X_B^{Q_B-\{1,k\}}$, by definition $(Q_z - \{p(z)\}) \cap (Q_B - \{p(B)\}) = Q_B - \{1,k\}$. Note that $p(B) = k$ (i.e., $R$). Since $P$ is a full path, $Q_z - \{p(z)\} = Q_x - \{p(x)\}$ and thus $(Q_x - \{p(x)\}) \cap (Q_B - \{k\}) = Q_B - \{1,k\}$. By Claim 4.6, for every subset $S$ of $Q_x - \{p(x)\}$, there exists vertex $r_S \in N_F[x] \cap T_x$ such that $(Q_{r_S} - \{p(r_S)\}) \cap (Q_x - \{p(x)\}) = S$. Let $\mathcal{S} = \{r_S : S \subseteq (Q_x - \{p(x)\}) \cap (Q_B - \{k\})\}$.

From the construction of the $r_S$ vertices given in the proof of Claim 4.6 it is clear that all the vertices in $\mathcal{S} - \{x\}$ are not on the path $P$ and therefore are neither ancestors nor descendants of $y$. By Fact 4.2 all these vertices are not adjacent to $y$ and thus must be in $V_a$ since they are adjacent to $x$ but not to $y$, and $x$ and $y$ have the same label at $a$. We have shown that all vertices of $\mathcal{S}$ must be in $V_a$.

Now suppose that fewer than $\lfloor k/2 \rfloor$ vertices of $Q_B - \{1,k\} = (Q_x - \{p(x)\}) \cap (Q_B - \{k\})$ are in $V_a$. Thus there are at least $\lfloor (k-1)/2 \rfloor$ vertices of $(Q_x - \{p(x)\}) \cap (Q_B - \{k\})$ that are outside $V_a$. Let $W'$ denote the set of these vertices and let $\mathcal{W} = \{r_S : S \subseteq W'\}$. Since $\mathcal{W} \subseteq \mathcal{S}$ we have shown above that all the vertices of $\mathcal{W}$ are in $V_a$. For every $S \subseteq W'$ the vertex $r_S$ is in $V_a$ and is adjacent to all the vertices of $S$ and to no other vertices in $W'$. Thus by Claim 4.10, all the vertices of $\mathcal{W}$ must have different labels at $a$. Now at least $|\mathcal{W}| \geq 2^{\lfloor (k-1)/2 \rfloor}$ different labels are needed at $a$. Since $2^{\lfloor (k-1)/2 \rfloor} \geq 2^{\lfloor k/2 \rfloor -1}$ we have contradicted $cwd(F) < 2^{\lfloor k/2 \rfloor -1}$.

Let $W$ denote the subset of $Q_B - \{1,k\}$ that is in $V_a$, where $|W| \geq \lfloor k/2 \rfloor$. We now show that all vertices in $W$ have unique labels at $a$. By Claim 4.11, there exists vertex $u \in X_B^{Q_B-\{1,k\}}$ such that $T_u$ is empty with respect to $a$. By Claim 4.6, for each vertex $w \in W$ there is a vertex $r_S$ in $T_u$ corresponding to the set $S = \{w\}$, such that $r_S$ is adjacent to $w$ and to no other vertex in $W$. Since $r_S$ is outside $V_a$, the label of the vertex $w$ at $a$ must be different from the labels at $a$ of all the other vertices of $W$. $\square$

We now prepare for the general induction step, each execution of which will add one more vertex requiring a new unique label. Thus executing the induction step $2^{\lfloor k/2 \rfloor -1} - \lfloor k/2 \rfloor$ times will complete the proof of the theorem. First we let $W$ denote a subset of $Q_B - \{1,k\}$ such that all elements of $W$ are in $V_a$ and $|W| = \lfloor k/2 \rfloor$. (Note that there may be more elements of $Q_B - \{1,k\}$ that are in $V_a$, but we only consider $\lfloor k/2 \rfloor$ of them.) By Claim 4.12 the set $W$ exists and all the vertices of $W$ have different labels at $a$.

Let $u$ be a vertex in $X_B^{Q_B - \{1,k\}}$ where $T_u$ is empty with respect to $a$. (Vertex $u$ is guaranteed by Claim 4.11.) Let $w_1$ and $w_2$ be arbitrary vertices in $W$. (Note that we require $k$ to be at least four in order to guarantee the existence of $w_1$ and $w_2$.) By Claim 4.7, $T_u$ contains vertices $u_1$ and $u_2$, at level at most $2k+1$ of $T_u$, such that $u_1$ is neither an ancestor nor a descendant of $u_2$ and

- $Q_{u_i} - \{p(u_i)\} = W_{u_i} \cup Y_{u_i}$, $i = 1, 2$, where $W_{u_i} = W - \{w_i\}$, $Y_{u_i} \cap W = \emptyset$, and $Y_{u_1} \cap Y_{u_2} = \emptyset$.

Note that $|W_{u_i}| = \lfloor k/2 \rfloor - 1$. Let $u'_i$ be an arbitrary element in $Y_{u_i}$ for $i = 1, 2$ and set $U = \{u_1, u_2\}$. Clearly, $W = \bigcup\{W_{u_i} : u_i \in U\}$. Let $P_U = \bigcup\{P_{u_i}^{Q_{u_i} - \{u'_i, p(u_i)\}} : u_i \in U\}$ and define $b$ to be a lowest vertex in $\mathcal{T}$ such that some path in $P_U$ is full with respect to $b$.

Note that since at least one path of $P_U$ must be in $V_b$ and all these paths are in $T_u$ and thus are not in $V_a$, we conclude that $b$ is not a descendant of $a$.

Claim 4.13. *$b$ is an ancestor of $a$.*

*Proof.* Suppose not and suppose $z \in X_{u_1}^{Q_{u_1} - \{u'_1, p(u_1)\}}$ is the root of a full path $P$ with respect to $b$, where $b$ is not an ancestor of $a$. Since such a path has more than $2^{\lfloor k/2 \rfloor}$ vertices, at least two vertices $x$ and $y$ on this path must have the same label at $b$. Suppose $x$ is closer to $z$ than is $y$. Since $P$ is a full path, $Q_z - \{p(z)\} = Q_y - \{p(y)\}$ and thus $W_{u_1} \subset Q_y - \{p(y)\}$. By Claim 4.6, for every subset $S$ of $W_{u_1}$, there exists a vertex $r_S \in N_F[y] \cap T_y$ such that $(Q_{r_S} - \{p(r_S)\}) \cap (Q_y - \{p(y)\}) = S$. Let $\mathcal{S} = \{r_S : S \subseteq W_{u_1}\}$ denote the set of all these vertices. From the construction of the $r_S$ vertices given in the proof of Claim 4.6 it is clear that all the vertices in $\mathcal{S} - \{y\}$ are adjacent to $y$ and are not adjacent to $p(y)$. We claim that all these vertices are not adjacent to $x$. If $x$ is the parent of $y$, then all these vertices are not adjacent to $x$, since $x = p(y)$ in this case. If $x$ is not the parent of $y$, then by Fact 4.8, $x$ and $y$ are not adjacent. Since all vertices of $\mathcal{S} - \{y\}$ are adjacent to $y$ and $y$ is not adjacent to $x$, it follows by Fact 4.4 that all these vertices are not adjacent to $x$. Thus, all vertices of $\mathcal{S} - \{y\}$ must be in $V_b$, since they are adjacent to $y$ but not to $x$. By definition, $y$ is in $V_b$ too. We have shown that all vertices of $\mathcal{S}$ must be in $V_b$.

By definition, all the vertices of $W_{u_1}$ are in $V_a$. Since we assume that $b$ is not an ancestor of $a$ and since $b$ cannot be a descendant of $a$ (as noted above) we see that all these vertices are not in $V_b$. For every $S \subseteq W_{u_1}$ there is a vertex $r_S$ in $V_b$ that is adjacent to all the vertices of $S$ and to no other vertices in $W_{u_1}$, and thus by Claim 4.10 all the vertices of $\mathcal{S}$ must have different labels at $b$. Since $|W_{u_1}| = \lfloor k/2 \rfloor - 1$, $|\mathcal{S}| = 2^{\lfloor k/2 \rfloor - 1}$, contradicting $cwd(F) < 2^{\lfloor k/2 \rfloor - 1}$. ∎

Claim 4.14. *For all $u_i \in U$, there is a vertex $y \in X_{u_i}^{Q_{u_i} - \{u'_i, p(u_i)\}}$ such that $T_y$ is empty with respect to $b$.*

*Proof.* If there is a path in $P_{u_i}^{Q_{u_i} - \{u'_i, p(u_i)\}}$ that is full with respect to $b$, this follows immediately from Claim 4.11. Otherwise, all trees rooted at vertices in $X_{u_i}^{Q_{u_i} - \{u'_i, p(u_i)\}}$ are mixed in the sense that some vertices are in $V_b$ and some are not. As in the proof of Claim 4.11, each such tree requires at least one label and these labels must all be different. Now, by Claim 4.9, the number of such trees is at least $2^{k+1}$, contradicting $cwd(F) < 2^{\lfloor k/2 \rfloor - 1}$. ∎

Claim 4.15. *All $W$ vertices have different labels at $b$.*

*Proof.* By the definitions, for all $w \in W$, there exists $u_i \in U$ such that $w \in W_{u_i}$. Now consider two arbitrary vertices $w', w''$ in $W$, and suppose $w' \in W_{u_1}$. (We will show that $w'$ and $w''$ must have different labels at $b$.) Let $y$ be a vertex in $X_{u_1}^{Q_{u_1}-\{u_1',p(u_1)\}}$ such that $T_y$ is empty with respect to $b$ (guaranteed by Claim 4.14). By definition, $(Q_y - \{p(y)\}) \cap (Q_{u_1} - \{p(u_1)\}) = Q_{u_1} - \{u_1', p(u_1)\}$. Since $w' \in Q_{u_1} - \{p(u_1)\}$ and $w'$ is not equal to $u_1'$, we get from the last formula that $w' \in Q_y - \{p(y)\}$.

Suppose $w'' \in Q_y - \{p(y)\}$. By Claim 4.6, there is a vertex $r_S$ in $T_y$ corresponding to the set $S = \{w'\}$, such that $r_S$ is adjacent to $w'$ and to no other vertex in $Q_y - \{p(y)\}$. Since $r_S$ is outside $V_b$ and is adjacent to $w'$ but not to $w''$, $w'$ and $w''$ must have different labels at $b$.

Suppose $w'' \notin Q_y - \{p(y)\}$. Again, let $r_S$ be a vertex (guaranteed by Claim 4.6) in $T_y$ corresponding to the set $S = \{w'\}$, such that $r_S$ is adjacent to $w'$ and to no other vertex in $Q_y - \{p(y)\}$. From the definition of $r_S$ in the proof of Claim 4.6 it is clear that except for $w'$, $r_S$ is adjacent just to vertices in $T_y$ and thus $r_S$ is not adjacent to $w''$. Since $r_S$ is outside $V_b$ and is adjacent to $w'$ but not to $w''$, $w'$ and $w''$ must have different labels at $b$. □

Let $u_1 \in U$ satisfy the existence of a vertex $z \in X_{u_1}^{Q_{u_1}-\{u_1',p(u_1)\}}$ such that $z$ is the root of a full path with respect to $b$.

CLAIM 4.16. *There are at least $\lfloor k/2 \rfloor$ vertices of $Q_{u_1} - \{u_1', p(u_1)\}$ that are in $V_b$.*

*Proof.* This proof is similar to that of Claim 4.12. As mentioned above, we assume that there is a full path $P$ with respect to $b$ rooted at $z \in X_{u_1}^{Q_{u_1}-\{u_1',p(u_1)\}}$. Since the path $P$ has more than $2^{\lfloor k/2 \rfloor}$ vertices, at least two vertices $x$ and $y$ on the path $P$ must have the same label at $b$. Suppose $x$ is closer to $z$ than is $y$. Since $z \in X_{u_1}^{Q_{u_1}-\{u_1',p(u_1)\}}$, by definition $(Q_z - p(z)) \cap (Q_{u_1} - p(u_1)) = Q_{u_1} - \{u_1', p(u_1)\}$. Since $P$ is a full path, $Q_z - \{p(z)\} = Q_x - \{p(x)\}$ and thus $(Q_x - p(x)) \cap (Q_{u_1} - p(u_1)) = Q_{u_1} - \{u_1', p(u_1)\}$. By Claim 4.6, for every subset $S$ of $Q_x - \{p(x)\}$, there exists vertex $r_S \in N_F[x] \cap T_x$ such that $(Q_{r_S} - \{p(r_S)\}) \cap (Q_x - \{p(x)\}) = S$. Let $\mathcal{S} = \{r_S : S \subseteq Q_{u_1} - \{u_1', p(u_1)\}$.

From the construction of the $r_S$ vertices given in the proof of Claim 4.6 it is clear that all the vertices in $\mathcal{S} - \{x\}$ are not on the path $P$ and therefore are neither ancestors nor descendants of $y$. By Fact 4.2 all these vertices are not adjacent to $y$ and thus must be in $V_b$ since they are adjacent to $x$ but not to $y$, and $x$ and $y$ have the same label at $b$. We have shown that all vertices of $\mathcal{S}$ must be in $V_b$.

Now suppose that fewer than $\lfloor k/2 \rfloor$ vertices of $Q_{u_1} - \{u_1', p(u_1)\}$ are in $V_b$. Thus there are at least $\lfloor (k-1)/2 \rfloor$ vertices of $Q_{u_1} - \{u_1', p(u_1)\}$ that are outside $V_b$. Let $M'$ denote the set of these vertices and let $\mathcal{M} = \{r_S : S \subseteq M'\}$. Since $\mathcal{M} \subseteq \mathcal{S}$, we have shown above that all the vertices of $\mathcal{M}$ are in $V_b$. For every set of vertices $S \subseteq M'$ the vertex $r_S$ is in $V_b$ and is adjacent to all the vertices of $S$ and to no other vertices in $M'$. Thus by Claim 4.10 all the vertices of $\mathcal{M}$ must have different labels at $b$. Thus at least $|\mathcal{M}| \geq 2^{\lfloor (k-1)/2 \rfloor}$ different labels are needed at $b$. Since $2^{\lfloor (k-1)/2 \rfloor} \geq 2^{\lfloor k/2 \rfloor - 1}$, we have contradicted $cwd(F) < 2^{\lfloor k/2 \rfloor - 1}$.

Let $M$ denote the subset of $Q_{u_1} - \{u_1', p(u_1)\}$ that is in $V_b$ where $|M| \geq \lfloor k/2 \rfloor$. We now show that all vertices in $M$ have unique labels at $b$. By Claim 4.11, there exists vertex $s \in X_{u_1}^{Q_{u_1}-\{u_1',p(u_1)\}}$ such that $T_s$ is empty with respect to $b$. By Claim 4.6, for each vertex $m \in M$ there is a vertex $r_S$ in $T_s$ corresponding to the set $S = \{m\}$, such that $r_S$ is adjacent to $m$ and to no other vertex in $M$. Since $r_S$ is outside $V_b$, the label of the vertex $m$ at $b$ must be different from the labels at $b$ of all the other vertices of $M$. □

Since $|W_{u_1}| = \lfloor k/2 \rfloor - 1$, there is a vertex $\widetilde{u_1} \in Y_{u_1} - \{u_1'\}$ that is in $V_b$. We now show that at $b$, $\widetilde{u_1}$ has a different label from all the vertices in $W$.

CLAIM 4.17. *At $b$, the label of $\widetilde{u_1}$ is different from the labels needed for the $W$ vertices.*

*Proof.* Suppose $\widetilde{u_1}$ and $w \in W$ have the same label at $b$. By Claim 4.14, there is a $y \in X_{u_1}^{Q_{u_1}-\{u'_1,p(u_1)\}}$ such that $T_y$ is empty with respect to $b$. By the definition of $y$, $Q_{u_1} - \{u'_1, p(u_1)\} \subset Q_y - \{p(y)\}$ and thus $\widetilde{u_1} \in Q_y - \{p(y)\}$. By Claim 4.6, there is a vertex $r_S$ in $T_y$ corresponding to the set $S = \{\widetilde{u_1}\}$, such that $r_S$ is adjacent to $\widetilde{u_1}$ and to no other vertex in $Q_y - \{p(y)\}$. Now, as in the proof of Claim 4.15, regardless of whether $w \in Q_y - \{p(y)\}$, it follows that $r_S$ is adjacent to $\widetilde{u_1}$ but not to $w$. Since $r_S$ is outside $V_b$, it follows that $\widetilde{u_1}$ and $w$ must have different labels at $b$. ∎

Thus this step has shown that at least $\lfloor k/2 \rfloor + 1$ distinct labels are required at $b$. To continue the process, we will augment $W$ with $\widetilde{u_1}$ and we will augment the set $U$ (as shown below) and define $c$ to be a lowest vertex in $\mathcal{T}$ such that some path in $P_U$ is full with respect to $c$. It will then be shown that $c$ is an ancestor of $b$ and that all vertices in $W$ must have different labels at $c$. A new vertex will be identified that must have a different label at $c$ from all vertices in $W$, and it will be added to $W$. This process continues by renaming $c$ to be $b$ and augmenting $U$.

From now on we assume that $W$ is augmented with $\widetilde{u_1}$. (Note that $W_{u_1}$ and $W_{u_2}$ are not changed.) Thus, the size of $W$ is now $\lfloor k/2 \rfloor + 1$. For augmenting $U$ we shall use the two vertices $u_{1,1}$ and $u_{1,2}$ guaranteed by the following claim.

CLAIM 4.18. *Let $\widetilde{w}$ be an arbitrary vertex in $W_{u_1}$. Then there exist vertices $u_{1,1}$ and $u_{1,2}$ in $T_{u_1}$ such that $u_{1,1}$ is neither an ancestor nor a descendant of $u_{1,2}$ and*

- *$Q_{u_{1,1}} - \{p(u_{1,1})\} = W_{u_{1,1}} \cup Y_{u_{1,1}}$, where $W_{u_{1,1}} = W_{u_1}$, and $Y_{u_{1,1}} \cap W = \emptyset$;*
- *$Q_{u_{1,2}} - \{p(u_{1,2})\} = W_{u_{1,2}} \cup Y_{u_{1,2}}$, where $W_{u_{1,2}} = W_{u_1} \cup \{\widetilde{u_1}\} - \{\widetilde{w}\}$, $Y_{u_{1,2}} \cap W = \emptyset$, $Y_{u_{1,1}} \cap Y_{u_{1,2}} = \emptyset$, and no vertex of $Y_{u_{1,1}}$ is an ancestor or a descendant of a vertex in $Y_{u_{1,2}}$.*

*Proof.* Using Claim 4.7 for $u = u_1$, $W = W_{u_1} \cup \{\widetilde{u_1}\}$, $w_1 = \widetilde{u_1}$, and $w_2 = \widetilde{w}$, there exist $u_{1,1}$ and $u_{1,2}$ in $T_{u_1}$ such that $u_{1,1}$ is neither an ancestor nor a descendant of $u_{1,2}$ and

- $Q_{u_{1,1}} - \{p(u_{1,1})\} = W_{u_{1,1}} \cup Y_{u_{1,1}}$, where $W_{u_{1,1}} = W_{u_1}$, $Y_{u_{1,1}} \cap (W_{u_1} \cup \{\widetilde{u_1}\}) = \emptyset$, and $Y_{u_{1,1}}$ is in $T_{u_1}$;
- $Q_{u_{1,2}} - \{p(u_{1,2})\} = W_{u_{1,2}} \cup Y_{u_{1,2}}$, where $W_{u_{1,2}} = W_{u_1} \cup \{\widetilde{u_1}\} - \{\widetilde{w}\}$, $Y_{u_{1,2}} \cap (W_{u_1} \cup \{\widetilde{u_1}\}) = \emptyset$, $Y_{u_{1,2}}$ is in $T_{u_1}$, $Y_{u_{1,1}} \cap Y_{u_{1,2}} = \emptyset$, and no vertex of $Y_{u_{1,1}}$ is an ancestor or a descendant of a vertex in $Y_{u_{1,2}}$.

To complete the proof of the claim we need to show that $Y_{u_{1,i}} \cap W = \emptyset$, $i = 1, 2$. We shall prove it for $i = 1$ (the case for $i = 2$ is the same). Suppose not, and let $t$ be a vertex in $Y_{u_{1,1}} \cap W$. Since $Y_{u_{1,1}} \cap (W_{u_1} \cup \{\widetilde{u_1}\}) = \emptyset$, it follows that $t$ is in $W - (W_{u_1} \cup \{\widetilde{u_1}\})$. By definition of $W$, $t \in W_{u_a}$ for some vertex $u_a$ in $U$ other than $u_1$. (Note that at the beginning there are just $u_1$ and $u_2$ in $U$ and thus in this case $u_a = u_2$. However, we use $u_a$ rather than $u_2$ so that the proof can be also applied for the more general case when the size of $U$ is greater than two.) Thus, $t$ is an ancestor of $u_a$. Now since $t \in Y_{u_{1,1}}$ and all vertices of $Y_{u_{1,1}}$ are in $T_{u_1}$, it follows that $t$ is a descendant of $u_1$. We conclude that $u_a$ is a descendant of $u_1$, a contradiction since no two vertices of $U$ are descendants of each other. ∎

*Augmenting $U$.* Vertex $u_1$ (a vertex in $U$ such that there is vertex $z \in X_{u_1}^{Q_{u_1}-\{u'_1,p(u_1)\}}$, where $z$ is the root of a full path with respect to $b$) will be replaced by the vertices $u_{1,1}$ and $u_{1,2}$ satisfying the conditions of Claim 4.18. As before, we let $u'_{1,1}$ and $u'_{1,2}$ be arbitrary elements in $Y_{u_{1,1}}$ and $Y_{u_{1,2}}$, respectively. After reindexing the elements of $U$ to be $u_1, u_2, \dots$, we define

- $P_U = \bigcup\{P_{u_i}^{Q_{u_i}-\{u'_i,p(u_i)\}} : u_i \in U\}$.

We now establish various claims about $U$.

CLAIM 4.19. *The following hold for $U$:*

1. *No vertex in $U$ is an ancestor (in $T$) of any other vertex in $U$.*
2. *For every vertex $u$ in $U$, we have $Q_u - \{p(u)\} = W_u \cup Y_u$, where $|W_u| = \lfloor k/2 \rfloor - 1$, $W_u \subset W$, and $Y_u \cap W = \emptyset$.*
3. *Set $W$ equals $\bigcup\{W_u : u \in U\}$.*
4. *For every pair $u, s$ of distinct vertices in $U$, $Y_u \cap Y_s = \emptyset$, and there is no vertex of $Y_u$ that is an ancestor or a descendant of a vertex of $Y_s$.*
5. *For every vertex $u$ in $U$, there is a vertex $y \in P_u^{Q_u - \{u', p(u)\}}$ such that $T_y$ is empty with respect to $b$.*
6. *All vertices in $W$ have different labels at $b$.*

*Proof.*

1. This follows by an easy induction argument and the fact that the new vertices $u_{1,1}$ and $u_{1,2}$ are descendants of $u_1$ and are not ancestors of each other.
2. Again we use an easy induction argument and the fact that the new vertices $u_{1,1}$ and $u_{1,2}$ satisfy the conditions of Claim 4.18.
3. This follows by an induction argument where it is assumed to be true for the unaugmented $U$ and $W$ (i.e., before $u_1$ was replaced by $u_{1,1}$ and $u_{1,2}$ and before $W$ was augmented with $\widetilde{u_1}$). Let $u_{1,1}$ and $u_{1,2}$ be the two new vertices replacing $u_1$. From the formula of Claim 4.18, it is clear that $W_{u_{1,1}} \cup W_{u_{1,2}} = W_{u_1} \cup \widetilde{u_1}$. Thus, replacing $W_{u_1}$ with $W_{u_{1,1}} \cup W_{u_{1,2}}$ in the formula $\bigcup\{W_u : u \in U\}$, we see that this formula is now equal to the augmented $W$.
4. Here the induction argument assumes it to be true for the unaugmented $U$. Let $u_{1,1}$ and $u_{1,2}$ be the two new vertices replacing $u_1$. If neither $u$ nor $s$ is in $\{u_{1,1}, u_{1,2}\}$, then the claim follows by the induction hypothesis. If both $u$ and $s$ are in $\{u_{1,1}, u_{1,2}\}$, then the claim follows from the formula of Claim 4.18. Suppose $u$ is in $\{u_{1,1}, u_{1,2}\}$ (say $u$ is equal to $u_{1,1}$) and $s$ is not in this set. By the formula of Claim 4.18, all vertices of $Y_{u_{1,1}}$ are in $T_{u_1}$, which implies that they are all descendants of $Y_{u_1}$. By the induction hypothesis, no vertex of $Y_{u_1}$ is an ancestor or a descendant of a vertex in $Y_s$. Thus, no vertex of $Y_{u_{1,1}} = Y_u$ is an ancestor or a descendant of a vertex in $Y_s$. This also implies that $Y_u \cap Y_s = \emptyset$.
5. For $u$ being neither $u_{1,1}$ nor $u_{1,2}$, this follows from Claim 4.14. For vertices $u_{1,1}$ or $u_{1,2}$, an argument almost identical to that used in the proof of Claim 4.11 suffices.
6. Immediate from Claims 4.15 and 4.17. $\square$

We now define $c$ to be a lowest vertex in $\mathcal{T}$ such that some path in $P_U$ is full with respect to $c$. Note that $c$ could be $b$, but from the definition of $c$ it is clear that $c$ cannot be a proper descendant of $b$. In order for the process to continue, the following analogues to Claims 4.13, 4.14, 4.15, 4.16, and 4.17 must hold. In all cases the proofs are the same as the proofs of the analogous claim and where appropriate use the facts established in Claim 4.19.

CLAIM 4.20 (analogue of Claim 4.13). *Vertex $c$ is an ancestor of $b$ (possibly a trivial ancestor of $b$).*

*Proof.* Suppose not and suppose $z \in X_{u_1}^{Q_{u_1} - \{u_1', p(u_1)\}}$ is the root of a full path $P$ with respect to $c$, where $c$ is not an ancestor of $b$. Since such a path has more than $2^{\lfloor k/2 \rfloor}$ vertices, at least two vertices $x$ and $y$ on this path must have the same label at $c$. Suppose $x$ is closer to $z$ than is $y$. Since $P$ is a full path, $Q_z - \{p(z)\} = Q_y - \{p(y)\}$ and thus $W_{u_1} \subset Q_y - \{p(y)\}$. By Claim 4.6, for every subset $S$ of $W_{u_1}$, there exists vertex

$r_S \in N_F[y] \cap T_y$ such that $(Q_{r_S} - \{p(r_S)\}) \cap (Q_y - \{p(y)\}) = S$. Let $\mathcal{S} = \{r_S : S \subseteq W_{u_1}\}$ denote the set of all these vertices. Using the same argument as in the proof of Claim 4.13, we conclude that all vertices of $\mathcal{S} - \{y\}$ must be in $V_c$, since they are adjacent to $y$ but not to $x$. Thus all vertices of $\mathcal{S}$ must be in $V_c$.

By definition, all the vertices of $W_{u_1}$ are in $V_b$. Since we assume that $c$ is not an ancestor of $b$ and since $c$ cannot be a proper descendant of $b$ (as noted above) we see that all these vertices are not in $V_c$. For every $S \subseteq W_{u_1}$ there is a vertex $r_S$ in $V_c$ that is adjacent to all the vertices of $S$ and to no other vertices in $W_{u_1}$, and thus by Claim 4.10, all the vertices of $\mathcal{S}$ must have different labels at $c$. By Claim 4.19 $|W_{u_1}| = \lfloor k/2 \rfloor - 1$. Thus, $|\mathcal{S}| = 2^{\lfloor k/2 \rfloor - 1}$, contradicting $cwd(F) < 2^{\lfloor k/2 \rfloor - 1}$. ∎

CLAIM 4.21 (analogue of Claim 4.14). *For all $u_i \in U$, there is a vertex $y \in X_{u_i}^{Q_{u_i} - \{u_i', p(u_i)\}}$ such that $T_y$ is empty with respect to $c$.*

*Proof.* The proof of this claim is obtained by rewriting the proof of Claim 4.14 replacing $b$ with $c$ everywhere. ∎

CLAIM 4.22 (analogue of Claim 4.15). *All $W$ vertices have different labels at $c$.*

*Proof.* By Claim 4.19, for all $w \in W$, there exists $u_i \in U$ such that $w \in W_{u_i}$. Now consider $w', w''$ two arbitrary vertices in $W$, and suppose $w' \in W_{u_1}$. Let $y$ be a vertex in $X_{u_1}^{Q_{u_1} - \{u_1', p(u_1)\}}$ such that $T_y$ is empty with respect to $c$ (guaranteed by Claim 4.21). The proof now continues as in the proof of Claim 4.15, replacing $b$ with $c$ everywhere to conclude that $w'$ and $w''$ must have different labels at $c$. ∎

As before, we let vertex $u_1 \in U$ satisfy the existence of a vertex $z \in X_{u_1}^{Q_{u_1} - \{u_1', p(u_1)\}}$, such that $z$ is the root of a full path with respect to $c$.

CLAIM 4.23 (analogue of Claim 4.16). *At least $\lfloor k/2 \rfloor$ vertices of $Q_{u_1} - \{u_1', p(u_1)\}$ are in $V_c$.*

*Proof.* The proof of this claim is obtained by rewriting the proof of Claim 4.16, replacing $b$ with $c$ everywhere. ∎

By Claim 4.19, $|W_{u_1}| = \lfloor k/2 \rfloor - 1$ and $Q_{u_1} - \{p(u_1)\} = W_{u_1} \cup Y_{u_1}$. Thus, by Claim 4.23, there is a vertex $\widetilde{u_1} \in Y_{u_1} - \{u_1'\}$ that is in $V_c$.

CLAIM 4.24 (analogue of Claim 4.17). *At $c$, the label of $\widetilde{u_1}$ is different from the labels needed for the $W$ vertices.*

*Proof.* Suppose $\widetilde{u_1}$ and $w \in W$ have the same label at $c$. By Claim 4.21, there is a $y \in X_{u_1}^{Q_{u_1} - \{u_1', p(u_1)\}}$ such that $T_y$ is empty with respect to $c$. By the definition of $y$, $Q_{u_1} - \{u_1', p(u_1)\} \subset Q_y - \{p(y)\}$ and thus $\widetilde{u_1} \in Q_y - \{p(y)\}$. By Claim 4.6, there is a vertex $r_S$ in $T_y$ corresponding to the set $S = \{\widetilde{u_1}\}$, such that $r_S$ is adjacent to $\widetilde{u_1}$ and to no other vertex in $Q_y - \{p(y)\}$. Now, as in the proof of Claim 4.22, regardless of whether $w \in Q_y - \{p(y)\}$, it follows that $r_S$ is adjacent to $\widetilde{u_1}$ but not to $w$. Since $r_S$ is outside $V_c$, it follows that $\widetilde{u_1}$ and $w$ must have different labels at $c$. ∎

Finally, we rename $c$ to be $b$ and again augment $W$ by adding $\widetilde{u_1}$ to $W$ and augment $U$ by replacing $u_1$ with the vertices $u_{1,1}$ and $u_{1,2}$ guaranteed by Claim 4.18. This augmentation continues until $|W| > 2^{\lfloor k/2 \rfloor - 1}$, thereby completing the proof of Theorem 1.3.

Note that these augmentation steps are based on the above claims where it is assumed that for every vertex $u$ in $U$, the depth of the tree $T_u$ is at least $2k+1+2^{\lfloor k/2 \rfloor}$. We now prove the correctness of this assumption. The vertex $u$ selected at the initial step is in $X_B^{Q_B - \{1,k\}}$ and thus is at level $2k+1$ of $T_B$. The set of vertices $U$ at the initial step is $\{u_1, u_2\}$, where, as indicated above, $u_1$ and $u_2$ are at level at most $2k+1$ of $T_u$. Thus, the vertices in the initial set $U$ are at level at most $2(2k+1)$ of $T_B$. By Claim 4.7, at each augmentation step the vertices $u_{1,1}$ and $u_{1,2}$ selected to replace $u_1$ in $U$ are at level at most $2k+1$ of $T_{u_1}$. Thus, after $2^{\lfloor k/2 \rfloor - 1}$ augmentation steps, all the vertices in $U$ are at level at most $(2k+1)2^{\lfloor k/2 \rfloor}$ of $T_B$. Now the claim follows since $\alpha$, the depth of $T$, is equal to $(2k+1)2^{\lfloor k/2 \rfloor + 1}$.

## 5. Concluding remarks.

The most important result in this paper is the fact that the clique-width of a graph can be exponentially higher than its treewidth. We fully expect that the bound expressed in Theorem 1.3 can be improved; the most interesting question, however, is to find a lower bound that confirms the bound achieved through some algorithm. In particular, we expect that the upper bound expressed in Theorem 1.2 is the best possible. Note that it is correct for $k$ equals one (trees) and for $k$ equals two (shown by a tedious exhaustive argument). Recently, Espelage, Gurski, and Wanke [9] have shown the existence of a linear time algorithm to determine whether a graph of bounded treewidth has clique-width at most $k$. Thus by using the result of Theorem 1.2 and running their algorithm for all values of $k$ from 1 to $3 * 2^{twd(G)-1}$, they have shown the existence of a linear time algorithm to determine the clique-width of a graph of bounded treewidth.

It is well appreciated that the most important open problems in the study of clique-width are the resolution of the general recognition problem (given graph $G$ and integer $k$, is $cwd(G) \leq k$?—strongly believed to be NP-complete) and the fixed $k$ recognition problem for $k \geq 4$. (The fixed $k$ recognition problem can be solved in polynomial time for $k \leq 3$; in particular the case when $k = 1$ is trivial, for $k = 2$ these are the cographs [8] which can be recognized in linear time [4], and for $k = 3$ an $O(n^2 m)$ algorithm is presented in [3].) It is interesting to note that the corresponding problems for treewidth are resolved. In particular, Arnborg, Corneil, and Proskurowski [1] showed that the general treewidth recognition problem (given graph $G$ and integer $k$, is $twd(G) \leq k$?) is NP-complete whereas the fixed recognition problem is in P. Bodlaender [2] later presented a linear time algorithm for the fixed $k$ treewidth recognition problem.

It seems as though the stumbling block for both problems is the difficulty in developing good arguments to provide strong lower bounds on the clique-width of a graph. Hopefully the techniques presented in section 4 can assist.

Another avenue of promising research is the development of polynomial time algorithms to determine the clique-width of restricted families of graphs, especially those where the clique-width can be arbitrarily large. Such families include permutation graphs, planar graphs, interval graphs, and even unit interval graphs. Again progress in this area depends on lower bound arguments that show that the clique-width achieved by a particular algorithm is the best possible.

**Acknowledgments.** The authors wish to thank the Natural Science and Engineering Research Council of Canada for their financial support and the Fields Institute at the University of Toronto for hosting their visits during this research. The thorough reading and helpful suggestions by the anonymous referee are greatly appreciated.

## REFERENCES

[1] S. Arnborg, D. G. Corneil, and A. Proskurowski, *Complexity of finding embeddings in a k-tree*, SIAM J. Algebraic Discrete Methods, 8 (1987), pp. 277–284.

[2] H. L. Bodlaender, *A linear-time algorithm for finding tree-decompositions of small treewidth*, SIAM J. Comput., 25 (1996), pp. 1305–1317.

[3] D. G. Corneil, M. Habib, J. M. Lanlignel, B. Reed, and U. Rotics, *Polynomial time recognition of clique-width $\leq 3$ graphs*, submitted for publication. An extended abstract has appeared in Proceedings of Latin American Theoretical Informatics, LATIN 2000, Lecture Notes in Comput. Sci. 1776, Springer-Verlag, Berlin, 2000, pp. 126–134.

[4] D. G. Corneil, Y. Perl, and L. K. Stewart, *A linear recognition algorithm for cographs*, SIAM J. Comput., 14 (1985), pp. 926–934.

[5] B. Courcelle, J. Engelfriet, and G. Rozenberg, *Handle-rewriting hypergraph grammars*, J. Comput. System Sci., 46 (1993), pp. 218–270.

[6] B. Courcelle, J. A. Makowsky, and U. Rotics, *Linear time solvable optimization problems on graphs of bounded clique-width*, Theory Comput. Syst., 33 (2000), pp. 125–150.

[7] B. Courcelle, J. A. Makowsky, and U. Rotics, *On the fixed parameter complexity of graph enumeration problems definable in monadic second order logic*, Discrete Appl. Math., 108 (2001), pp. 23–52.

[8] B. Courcelle and S. Olariu, *Upper bounds to the clique-width of graphs*, Discrete Appl. Math., 101 (2000), pp. 77–114.

[9] W. Espelage, F. Gurski, and E. Wanke, *Deciding clique-width for graphs of bounded treewidth*, J. Graph Algorithms Appl., 7 (2003), pp. 141–180.

[10] M. U. Gerber and D. Kobler, *Algorithms for vertex partitioning problems on graphs with fixed clique-width*, Theoret. Comput. Sci., 299 (2003), pp. 719–734.

[11] D. Kobler and U. Rotics, *Edge dominating set and colorings on graphs with fixed clique-width*, Discrete Appl. Math., 126 (2003), pp. 197–221.

[12] J. A. Makowsky and U. Rotics, *On the classes of graphs with few $P_4$'s*, Internat. J. Found. Comput. Sci., 10 (1999), pp. 329–348.

SIAM J. COMPUT. © 2005 Society for Industrial and Applied Mathematics
Vol. 34, No. 4, pp. 848–862

# WORK-COMPETITIVE SCHEDULING FOR COOPERATIVE COMPUTING WITH DYNAMIC GROUPS[*]

CHRYSSIS GEORGIOU[†], ALEXANDER RUSSELL[‡], AND ALEXANDER A. SHVARTSMAN[§]

**Abstract.** The problem of cooperatively performing a set of $t$ tasks in a decentralized computing environment subject to failures is one of the fundamental problems in distributed computing. The setting with partitionable networks is especially challenging, as algorithmic solutions must accommodate the possibility that groups of processors become disconnected (and, perhaps, reconnected) during the computation. The efficiency of task-performing algorithms is often assessed in terms of *work*: the total number of tasks, counting multiplicities, performed by all of the processors during the computation. In general, the scenario where the processors are partitioned into $g$ disconnected components causes any task-performing algorithm to have work $\Omega(t \cdot g)$ even if each group of processors performs no more than the optimal number of $\Theta(t)$ tasks.

Given that such pessimistic lower bounds apply to *any* scheduling algorithm, we pursue a *competitive* analysis. Specifically, this paper studies a simple randomized scheduling algorithm for $p$ asynchronous processors, connected by a dynamically changing communication medium, to complete $t$ known tasks. The performance of this algorithm is compared against that of an omniscient off-line algorithm with full knowledge of the future changes in the communication medium. The paper describes a notion of *computation width*, which associates a natural number with a history of changes in the communication medium, and shows both upper and lower bounds on work-competitiveness in terms of this quantity. Specifically, it is shown that the simple randomized algorithm obtains the competitive ratio $(1 + \mathbf{cw}/e)$, where $\mathbf{cw}$ is the computation width and $e$ is the base of the natural logarithm ($e = 2.7182\dots$); this competitive ratio is then shown to be tight.

**Key words.** on-line algorithms, competitive analysis, partitionable networks, distributed computation, independent tasks, randomized algorithms, work complexity

**AMS subject classifications.** 68W15, 68W20, 68W40, 68Q25, 68Q85

**DOI.** 10.1137/S0097539704440442

## 1. Introduction.

The problem of cooperatively performing a known set of tasks in a decentralized computing environment subject to failures is one of the fundamental problems in distributed computing. Variations on this problem have been studied in a variety of different settings, including, for example, message-passing models [7, 8, 11], shared-memory models [18, 17, 2, 21, 19], and partitionable network models [10, 20]. In the settings where network partitions may interfere with the progress of computation, the challenge is to maintain efficiency despite dynamically changing processor connectivity.

This problem is normally abstracted in terms of a set of $t$ tasks that must be performed in a distributed environment consisting of $p$ processors, subject to processor failures and communication disruptions. Algorithmic solutions for this problem are

[*]Received by the editors February 2, 2004; accepted for publication (in revised form) December 8, 2004; published electronically May 12, 2005. This research was supported in part by NSF grant 0311368.

http://www.siam.org/journals/sicomp/34-4/44044.html

[†]Department of Computer Science, University of Cyprus, Nicosia, Cyprus (chryssis@ucy.ac.cy). This work was performed in part while this author was at the University of Connecticut.

[‡]Department of Computer Science and Engineering, University of Connecticut, Storrs, CT 06269 (acr@cse.uconn.edu). The work of this author was supported in part by NSF Career Award 0093065 and by NSF grants 0220264 and 0218443.

[§]Department of Computer Science and Engineering, University of Connecticut, Storrs, CT 06269, and Laboratory for Computer Science and Artificial Intelligence, Massachusetts Institute of Technology, Cambridge, MA 02139 (alex@theory.csail.mit.edu). The work of this author was supported in part by NSF Career Award 9984778 and by NSF grants 9988304 and 0121277.

typically evaluated by bounding their worst-case *work*: the total number of computation steps performed by all processors during the computation. We consider the situation where the tasks are *similar*, that is, completion of each task requires the same number of computation steps, and where task-oriented work dominates local bookkeeping. In this case the work incurred by an algorithm is simply the total number of tasks, counting multiplicities, completed by the processors.

The details of the computation model naturally have a dramatic impact on the existence of efficient (or even interesting) algorithms for the problem. In this paper, we consider the *partitionable network* scenario consisting of $p$ asynchronous processors with a communication medium that is subject to arbitrary *partitions* during the life of the computation. This model is motivated by the abstraction provided by a typical *group communication scheme*; see, for example, the surveys in [23]. Specifically, at each point of the computation, the communication medium effectively partitions the processors into nonoverlapping *groups*: communication within a group is instantaneous and reliable, communication across groups is impossible. Naturally, processors in the same group can share their knowledge of completed tasks and, while they remain connected, avoid doing redundant work. For the remainder of the paper we refer to a transition from one network partition to another as a *reconfiguration*.

We do not charge for coordination within a group, simply treating grouped processors as a single (virtual) asynchronous processor. In particular, if a group of processors performs a set of $t$ tasks during the lifetime of that group, we charge this group $t$ units of work, ignoring, for example, partially completed tasks which may remain at the group's demise or the cost of synchronizing processors' knowledge during the group's inception. Each processor may cease executing tasks *only* when it knows the results of all tasks. While processors are asynchronous, they do not crash.

An algorithm in this model is a rule which, given a group of processors and a set of tasks known by this group to be completed, determines a task for the group to complete next. In the case where all processors are disconnected during the entire computation, any algorithm must incur $\Omega(t \cdot p)$ work. On the other hand, any reasonable algorithm should attain $O(t)$ work in the case where all processors remain connected during the computation. Considering that *every* algorithm performs poorly in the totally disconnected case, it seems reasonable to treat the problem as an on-line problem and pursue competitive analysis.

Fix, for the moment, an algorithm $A$. For expository purposes, let us treat both the processors' asynchrony and the dynamics of the network as if they were determined by an adversary $\mathcal{A}$. The adversary determines an initial partition $\mathcal{P}_1$ of the processors into groups and determines how many tasks each group of this partition $\mathcal{P}_1$ completes before the next reconfiguration; while the *number* of tasks completed by each group is determined by the adversary, the actual subset of tasks (that is, the *identity* of the tasks) completed by each group is determined by the algorithm $A$. The adversary then determines a reconfiguration of the processors, giving rise to a new partition $\mathcal{P}_2$, and, as before, determines how many tasks each of the newly created groups of $\mathcal{P}_2$ completes before the next reconfiguration. Any group created during such a reconfiguration is assumed to have the combined knowledge of all its members: any task known to be completed by a processor of the group $G$ is known to be completed by all processors of $G$. This process of reconfiguration and computation continues until every processor is aware of the outcome of every task. Groups with knowledge of the outcome of all tasks cause no work: in effect, they may "idle" until the next reconfiguration. Note that for this algorithm $A$, the work caused by the adversary $\mathcal{A}$ is completely determined by (i) the collection of groups that existed during the computation, (ii) the number of tasks $\mathcal{A}$ permits each group to perform, and (iii) for each group $G$, the identities of all those groups in which processors of $G$ have previously been members. (Note that the initial knowledge of the group $G$ is determined in part by (iii).) These characteristics can be captured by a certain directed acyclic graph, to which we refer as a *computation pattern.* This is formally defined in the next section. Note that different sequences of reconfigurations can in fact give rise to the same computation pattern.

As an example, consider the scenario with 3 processors which, starting from isolation, are permitted to proceed synchronously until each has completed $t/2$ tasks; at this point an adversary chooses a pair of processors to merge into a group. It is easy to show that if $T_1$, $T_2$, and $T_3$ are subsets of $[t]$ of size $t/2$, then there is a pair $(T_i, T_j)$ (where $i \neq j$) so that $|T_i \cap T_j| \geq t/6$: in particular, for *any* scheduling algorithm, there is a pair of processors which, if merged at this point, will have $t/6$ duplicated tasks; this pair alone must then expend $t + t/6$ work to complete all $t$ tasks. The optimal off-line algorithm that schedules tasks with full knowledge of future merges, of course, accrues only $t$ work for the merged pair, as it can arrange for zero overlap. Furthermore, if the adversary partitions the two merged processors immediately after the merge (after allowing the processors to exchanged information about task executions), then the work performed by the merged and then partitioned pair is $t + t/3$; the work performed by the optimal algorithm remains unchanged, since it terminates at the merge.

**Contributions.** We study upper and lower bounds on the competitiveness of scheduling algorithms for the task-performing problem in partitionable networks. We analyze the natural randomized algorithm for $p$ processors and $t$ tasks, called RANDOM SELECT (RS), in which each processor (or group) determines the next task to complete by randomly selecting the task from the subset of tasks this group does not know to be completed. We compare the expected work of this algorithm to the work of an optimal off-line algorithm, which may schedule tasks with full knowledge of future partitions.

In order to precisely state the results of the paper, we pause to introduce some notation. In the literature, groups of processors are given structured names, such that a group $G$ is a pair $\langle G.id, G.set\rangle$, where $G.id$ is the unique identifier of $G$ and $G.set$ is the set of processor identifiers in $[p]$ that determine the members of the group. To reduce notational clutter, given a group named $G$, we use $G$ to stand for $G.set$ in this paper (e.g., if two, possibly distinct, groups $G$ and $G'$ have identical membership, we express this by $G = G'$).

As discussed previously, an adversary determines a *computation pattern* $C$ in a natural way; this is a directed acyclic graph (DAG), each vertex corresponding to a group of processors that exists during some point of the computation; a directed edge is placed from group $G$ to group $G'$ if $G \cap G' \neq \emptyset$ and $G'$ was formed by a reconfiguration involving processors in $G$ (this is discussed and formally defined in section 2). We say that two groups $G$ and $G'$ are *independent* if there is no directed path connecting one to the other. For such a pattern $C$, the *computation width* of $C$, denoted $\mathbf{cw}(C)$, is the maximum number of independent groups reachable (along directed paths) in this DAG from any vertex. We show the following:

- (Upper bound.) For any computation pattern $C$, the randomized algorithm RS discussed above is $(1 + \mathbf{cw}(C)/e)$-work competitive.
- (Lower bound.) For any scheduling algorithm $A(p, t)$, any $\epsilon > 0$, and any

nonzero $k \in \mathbb{N}$, there exist $p$, $t$, and a computation pattern $C$ so that $\mathbf{cw}(C) = k$ and the work performed by algorithm $A(p,t)$ is at least $(1 + k/e - \epsilon)$ times that of the off-line algorithm.

In particular, RS achieves the *optimal* competitive ratio over the set of all computation patterns with a given computation width.

**Prior and related work; motivation.** The problem of distributed cooperation for message-passing models was introduced and studied by Dwork, Halpern, and Waarts [11], who defined the notion of (task-oriented) work. The current problem of cooperation in *partitionable* networks has been the subject of active research. However, known solutions address narrow special cases, or provide substantially weaker bounds. Dolev, Segala, and Shvartsman [10] performed the first study of the problem in the partitionable setting. They model reconfiguration patterns for which the termination time of any on-line task-performing algorithm is greater than the termination time of an off-line task-performing algorithm by a factor linear in $p$. Malewicz, Russell, and Shvartsman [20] introduced the notion of *h-waste* that measures the worst-case redundant work performed by $h$ groups (or processors) when started in isolation and merged into a single group at some later time. While these results are deterministic, they only adequately describe such computation to the point of the *first* reconfiguration, where the reconfiguration is further assumed to simply *merge* groups together. Georgiou and Shvartsman [16] give upper bounds on work for an algorithm that performs work in the presence of network fragmentations and merges (i.e., limited patterns of reconfigurations) using a group communication service where processors initially start in a single group. They establish an upper bound of $O(\min(t{\cdot}p,\ t{+}t{\cdot}g(C)))$, where $g(C)$ is the total number of new groups formed during the computation pattern $C$. Note that $\mathbf{cw}(C) \leq g(C)$, and there can be an arbitrary gap between $\mathbf{cw}(C)$ and $g(C)$.

Thus prior work established reasonably tight (in the length of the processor schedule) results for a *single first* merge [20], illustrated the fact that on-line algorithms subject to diverging reconfiguration patterns incur linear (in $p$) overhead relative to an off-line algorithm [10], and showed an upper bound for an algorithm using group communication services for a limited pattern of reconfigurations starting with a *single* group [16].

The problem of cooperation on a common set of tasks in distributed settings has been studied in message-passing models [7, 8, 11]. These studies present various load-balancing techniques for structuring the work for computing devices that are able to communicate by means of point-to-point messages. The studies of Georgiades, Mavronicolas, and Spirakis [13] and Papadimitriou and Yannakakis [22] investigated the impact of communication topology on the effectiveness of load-balancing.

The notion of competitiveness was introduced by Sleator and Tarjan [26] (see also Bartal, Fiat, and Rabani [5], Awerbuch, Kutten, and Peleg [3], and Ajtai et al. [1]).

Group communication services have become important as building blocks for fault-tolerant distributed systems. Such services enable processors located in a failure-prone network to operate collectively as a group, using the services to multicast messages to group members (see the special issue [23]). To evaluate the effectiveness of partitionable group communication services, Sussman and Marzullo [27] proposed a measure (*cushion*) precipitated by a simple partition-aware application. Babaoglu et al. [4] studied systematic support for partition awareness based on group communication services in a wide range of application areas. As we mentioned earlier, cooperation on a common set of tasks has also been studied for algorithms using group communications [10, 16].

A related problem, referred to as Write-All, has been studied in the shared-memory model. Early work in this area was reported by Kanellakis and Shvartsman [18], Martel and Subramonian [21], Kedem, Palem, and Spirakis [19], and Anderson and Woll [2]. In this setting the processors cooperate on updating locations in shared memory. The algorithmic techniques and analysis found there are quite different from the ones we present in this paper. Another related shared-memory problem, called *Collect*, requires that each processor learn the private values of all other processors. This problem was introduced by Shavit [25] and studied by Saks, Shavit, and Woll [24].

The structure of this paper is as follows. In section 2 we define the problem and model of computation. In section 3 we present and analyze the randomized algorithm RS. In section 4 we prove a lower bound for the problem. We conclude in section 5.

Abstracts describing preliminary versions of the results in this paper appear in [14, 15].

## 2. Model and definitions.

We consider a distributed system consisting of $p$ asynchronous processors connected by communication links; each processor has a unique identifier from the set $[p] = \{1, 2, \ldots, p\}$; the value $p$ is known to all processors. The problem is then defined in terms of $t$ tasks with unique identifiers, initially known to all processors. The tasks are independent and idempotent—multiple executions of the same task have the same effect as a single execution. Processors may cease executing tasks only when they know the results of all tasks. This general problem is often referred to as *Do-All*.

The model is complicated by subjecting the processors to dynamic changes in the communication medium. In particular, at each instant of time, the network is partitioned into a collection of *groups*. Communication between processors in the same group is instantaneous and reliable, so that grouped processors may perfectly cooperate to complete tasks; communication across groups, however, is not possible. We consider the dynamic case where communication can be arbitrarily lost and reestablished. In particular, the computation of the processors is punctuated by a sequence of *reconfigurations*; each reconfiguration may induce an arbitrary change in the partition of the processors into groups. We shall assume that task executions are atomic with respect to reconfigurations. That is, a reconfiguration does not occur when some tasks are "halfway" through execution.

In order to focus on scheduling issues, we assume that processors in a single group work as a single virtual unit; indeed, we will treat them as a single asynchronous processor. In particular, upon the establishment of a new group by a reconfiguration, the processors in the group share their knowledge (of completed tasks) before they continue processing. A deterministic algorithm $D$ in this model is a rule which, given a processor (or group of processors) and a collection of tasks known to be completed, determines the next task for this processor (or group) to complete. Specifically, an algorithm is a function $D : 2^{[p]} \times 2^{[t]} \to [t]$; we note that the lower bounds proved in this paper actually apply to a wider class of algorithms that may in fact take into account the entire history of the computation of the group in question. For simplicity, we assume that $\forall P \subset [p], \forall T \subsetneq [t], D(P, T) \notin T$, which is to say that the algorithm never chooses to complete a task it already knows to be completed. Our goal will be to design algorithms that schedule the execution of the tasks to minimize the total *work*, where work is defined to be *the number of tasks executed by all the processors during the entire computation (counting multiplicities).* Ideally, the sets of tasks completed by two groups of processors when these groups are merged should be disjoint to avoid wasted effort. This is impossible in general, as processors must schedule their work in ignorance of future reconfigurations and, moreover, circumstances where two groups of processors merge that have collectively completed more than $t$ tasks will necessitate wasted work. A processor may cease executing tasks only when it knows the results of all tasks. We refer to this version of the Do-All problem as *Omni-Do*.

We will consider the behavior of an algorithm in the face of an adversary (which is *oblivious* in the sense of [6]) that determines both the *sequence of reconfigurations* and the *number of tasks completed* by each group before it is involved in another reconfiguration. Taken together, this information determines a *computation pattern*: this is a DAG, each vertex of which corresponds to a group $G$ of processors that existed during the computation; a directed edge is placed from $G_1$ to $G_2$ if $G_2$ was created by a reconfiguration involving $G_1$. We label each vertex of the DAG with the group of processors associated with that vertex and the total number of tasks that the adversary allows the group of processors to perform before the next reconfiguration occurs. As mentioned before, different adversaries (causing different sequences of reconfigurations) may give rise to the same computation pattern; the *work* caused by an adversary, however, depends only on the computation pattern determined by that adversary.

Specifically, if $t$ is the number of tasks and $p$ the number of processors, then such a computation pattern is a labeled and weighted DAG, which we call a $(p,t)$-DAG.

DEFINITION 2.1. *A $(p,t)$-DAG is a DAG $C = (V,E)$ augmented with a weight function $h : V \to [t] \cup \{0\}$ and a labeling $g : V \to 2^{[p]} \setminus \{\emptyset\}$ so that the following hold.*

- *For any maximal path $P = (v_1, \ldots, v_k)$ in $C$, $\sum h(v_i) \geq t$. (This guarantees that any algorithm terminates during the computation described by the DAG.)*
- *$g$ possesses the following "initial conditions":*

$$[p] = \dot{\bigcup}_{v:\ in(v)=0} g(v).$$

- *$g$ respects the following "conservation law":*
  *there is a function $\phi : E \to 2^{[p]} \setminus \{\emptyset\}$ so that for each $v \in V$ with $\mathrm{in}(v) > 0$,*

$$g(v) = \dot{\bigcup}_{(u,v)\in E} \phi\big((u,v)\big),$$

  *and for each $v \in V$ with $\mathrm{out}(v) > 0$,*

$$g(v) = \dot{\bigcup}_{(v,u)\in E} \phi\big((v,u)\big).$$

In the above definition, $\dot{\cup}$ denotes disjoint union, and $\mathrm{in}(v)$ and $\mathrm{out}(v)$ denote the in-degree and out-degree of $v$, respectively. Finally, for two vertices $u, v \in V$, we write $u \leq v$ if there is a directed path from $u$ to $v$; we then write $u < v$ if $u \leq v$ and $u$ and $v$ are distinct.

*Example.* As an example, consider the $(12,t)$-DAG shown on Figure 2.1. Here we have $g_1 = \{p_1\}$, $g_2 = \{p_2, p_3, p_4\}$, $g_3 = \{p_5, p_6\}$, $g_4 = \{p_7\}$, $g_5 = \{p_8, p_9, p_{10}, p_{11}, p_{12}\}$, $g_6 = \{p_1, p_2, p_3, p_4, p_6\}$, $g_7 = \{p_8, p_{10}\}$, $g_8 = \{p_9, p_{11}, p_{12}\}$, $g_9 = \{p_1, p_2, p_3, p_4, p_6, p_8, p_{10}\}$, $g_{10} = \{p_5, p_{11}\}$, and $g_{11} = \{p_9, p_{12}\}$.

This computation pattern models all asynchronous computations (adversaries) with the following behavior: (i) The processors in groups $g_1$ and $g_2$ and processor $p_6$ of group $g_3$ are regrouped during some reconfiguration to form group $g_6$. Processor $p_5$ of

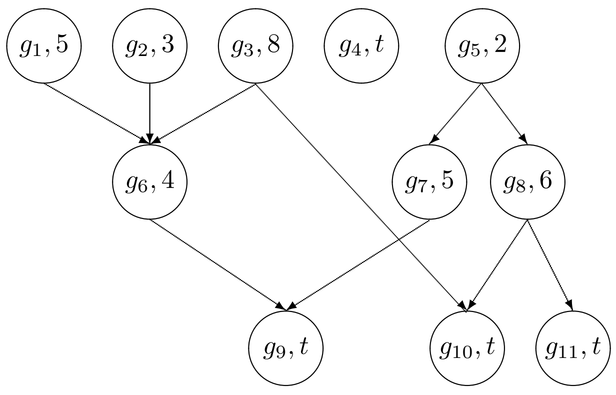


FIG. 2.1. *An example of a* $(12, t)$*-DAG.*

group $g_3$ becomes a member of group $g_{10}$ during the same reconfiguration (see below). Prior to this reconfiguration, processor $p_1$ (the singleton group $g_1$) has performed exactly 5 tasks, the processors in $g_2$ have cooperatively performed exactly 3 tasks, and the processors in $g_3$ have cooperatively performed exactly 8 tasks (assuming that $t > 8$). (ii) Group $g_5$ is partitioned during some reconfiguration into two new groups, $g_7$ and $g_8$. Prior to this reconfiguration, the processors in $g_5$ have performed exactly 2 tasks. (iii) Groups $g_6$ and $g_7$ merge during some reconfiguration and form group $g_9$. Prior to this merge, the processors in $g_6$ have performed exactly 4 tasks (counting only the ones performed after the formation of $g_6$ and assuming that there are at least 4 tasks remaining to be done) and the processors in $g_7$ have performed exactly 5 tasks. (iv) The processors in group $g_8$ and processor $p_5$ of group $g_3$ are regrouped during some reconfiguration into groups $g_{10}$ and $g_{11}$. Prior to this reconfiguration, the processors in group $g_8$ have performed exactly 6 tasks (assuming that there are at least 6 tasks remaining, otherwise they would have performed the remaining tasks). (v) The processors in $g_9$, $g_{10}$, and $g_{11}$ run until completion with no further reconfigurations. (vi) Processor $p_7$ (the singleton group $g_4$) runs in isolation for the entire computation.

Let $D$ be a deterministic algorithm for Omni-Do and $C$ a computation pattern. We then let $W_D(C)$ denote the total work expended by algorithm $D$, where reconfigurations are determined according to the computation pattern $C$. $W_D$ is formally defined as follows.

DEFINITION 2.2. *Let $C$ be a $(p, t)$-DAG and $D$ a deterministic algorithm for Omni-Do.* $\boldsymbol{W_D(C)}$ *is defined inductively as follows.*

- *For a vertex $v$ of $C$ with* $\text{in}(v) = 0$, *define $L_v$ to be the set containing the first $h(v)$ tasks completed by group $g(v)$ according to $D$.*
- *Otherwise,* $\text{in}(v) > 0$*; in this case, let $\check{L}_v = \bigcup_{u<v} L_u$ denote the collection of all tasks known to be complete at the inception of group $g(v)$. Then let $L_v$ be the first $h(v)$ tasks completed by group $g(v)$ according to $D$ starting with knowledge $\check{L}_v$. If $h(v) > t - |\check{L}_v|$, define $L_v = [t] \setminus \check{L}_v$.*

*Then* $W_D(C) = \sum_{v \in C} |L_v|$.

We treat randomized algorithms as distributions over deterministic algorithms; for a set $\Omega$ and a family of deterministic algorithms $\{D_r \mid r \in \Omega\}$ we let $R = \mathcal{R}(\{D_r \mid r \in \Omega\})$ denote the randomized algorithm where $r$ is selected uniformly at random from $\Omega$ and scheduling is done according to $D_r$. For a real-valued random variable $X$, we let $\mathbb{E}[X]$ denote its expected value. We let OPT denote the optimal (off-line) algorithm. Specifically, for each $C$ we define $W_{\mathrm{OPT}}(C) = \min_D W_D(C)$.

DEFINITION 2.3 (see [26, 12, 6]). *Let $\alpha$ be a real-valued function defined on the set of all $(p, t)$-DAGs ($\forall\, p$ and $t$). A randomized algorithm $R$ is $\alpha$-competitive if for all computation patterns $C$,*

$$\mathbb{E}[W_{D_r}(C)] \leq \alpha(C) W_{OPT}(C),$$

*this expectation being taken over uniform choice of $r \in \Omega$.*

Presently, we will introduce a function $\alpha$ that depends on a certain parameter (see Definition 2.7) of the graph structure of $C$. We note that, by definition, $\alpha \geq 1$.

We pause to develop some terminology that we will use in the rest of the paper.

DEFINITION 2.4. *A* partially ordered set, *or* poset, *is a pair $(P, \leq)$, where $P$ is a set and $\leq$ is a binary relation on $P$ for which* (i) *$\forall\, x \in P$, $x \leq x$;* (ii) *if $x \leq y$ and $y \leq x$, then $x = y$; and* (iii) *if $x \leq y$ and $y \leq z$, then $x \leq z$. For a poset $(P, \leq)$ we overload the symbol $P$, letting it denote both the set and the poset.*

DEFINITION 2.5. *Let $P$ be a poset. We say that two elements $x$ and $y$ of $P$ are* comparable *if $x \leq y$ or $y \leq x$; otherwise $x$ and $y$ are* incomparable. *A* chain *is a subset $H$ of $P$ such that any two elements of $H$ are comparable. An* antichain *is a subset $A$ of $P$ such that any two distinct elements of $A$ are incomparable. The* width *of $P$, denoted $\mathbf{w}(P)$, is the size of the largest antichain of $P$.*

Associated with any DAG $C = (V, E)$ is the natural *vertex poset* $(V, \leq)$, where $u \leq v$ if and only if there is a directed path from $u$ to $v$. Then the *width* of $C$, denoted $\mathbf{w}(C)$, is the width of the poset $(V, \leq)$.

DEFINITION 2.6. *Given a DAG $C = (V, E)$ and a vertex $v \in V$, we define the* predecessor graph at *$v$, denoted $P_C(v)$ (or $P(v)$ when $C$ is implied), to be the subgraph of $C$ that is formed by the union of all paths in $C$ terminating at $v$. Likewise, the* successor graph at *$v$, denoted $S_C(v)$ (or $S(v)$ when $C$ is implied), is the subgraph of $C$ that is formed by the union of all the paths in $C$ originating at $v$.*

DEFINITION 2.7. *The* computation width *of a DAG $C = (V, E)$, denoted $\mathbf{cw}(C)$, is defined as*

$$\mathbf{cw}(C) = \max_{v \in V} \mathbf{w}(S(v)).$$

Note that the processors that comprise a group formed during a computation pattern $C$ may be involved in many different groups at later stages of the computation, but no more than $\mathbf{cw}(C)$ of these groups will be forced to compute in ignorance of each other's progress.

In the $(12, t)$-DAG of Figure 2.1, the maximum width among all successor graphs is 3: $\mathbf{w}(S((g_5, 2))) = 3$. Hence, the computational width of this DAG is 3. Note that the width of the DAG is 6 (nodes $(g_1, 5)$, $(g_2, 3)$, $(g_3, 8)$, $(g_4, t)$, $(g_7, 5)$, and $(g_8, 6)$ form an antichain of maximum size).

## 3. Algorithm RS and its analysis.

In this section we present the RANDOM SELECT (RS) algorithm and its analysis.

### 3.1. Description of algorithm RS.

We consider the natural randomized algorithm RS, where a processor (or group), with knowledge that the tasks in a set $K \subset [t]$ have been completed, selects to next complete a task at random from the set $[t] \setminus K$. More formally, let $\Pi = (\pi_1, \ldots, \pi_p)$ be a $p$-tuple of permutations, where each $\pi_i$ is a permutation of $[t]$. We describe a deterministic algorithm $D_\Pi$ so that

$$\mathrm{RS} = \mathcal{R}\big(\{D_\Pi \mid \Pi \in (S_t)^p\}\big);$$

here $S_t$ is the collection of permutations on $[t]$. Let $G$ be a group of processors and $\gamma \in G$ the processor in $G$ with the lowest processor identifier. Then the deterministic algorithm $D_\Pi$ specifies that the group $G$, should it know that the tasks in $K \subset [t]$ have been completed, next completes the first task in the sequence $\pi_\gamma(1), \dots, \pi_\gamma(t)$ which is not in $K$.

**3.2. Analysis of algorithm RS.** We now analyze the competitive ratio (in terms of work) of algorithm RS. We write $W_{\text{RS}}(C) = \mathbb{E}\,[W_{\text{RS}}(C)]$, this expectation taken over the random choices of the algorithm. Where $C$ can be inferred from context, we simply write $W_{\text{RS}}$ and $W_{\text{OPT}}$.

We first recall Dilworth's lemma [9], a duality theorem for posets.

LEMMA 3.1 (see [9]). *The width of a poset $P$ is equal to the minimum number of chains needed to cover $P$. (A family of nonempty subsets of a given set $S$ is said to* cover *$S$ if their union is $S$.)*

We will also use a generalized degree-counting argument.

LEMMA 3.2. *Let $G = (U, V, E)$ be an undirected bipartite graph with no isolated vertices and $h : V \to \mathbb{R}$ a nonnegative weight function on $G$. For a vertex $v$, let $\Gamma(v)$ denote the vertices adjacent to $v$. Suppose that for some $A > 0$ and for every vertex $u \in U$ we have $\sum_{v\in\Gamma(u)} h(v) \le A$ and that for some $B > 0$ and for every vertex $v \in V$ we have $\sum_{u\in\Gamma(v)} h(u) \ge B$. Then*

$$\frac{\sum_{u\in U} h(u)}{\sum_{v\in V} h(v)} \ge \frac{B}{A}.$$

*Proof.* We compute the quantity $\sum_{(u,v)\in E} h(u)h(v)$ by expanding according to each side of the bipartition:

$$\begin{aligned} A\sum_{u\in U} h(u) &\ge \sum_{u\in U}\Big(h(u)\cdot\sum_{v\in\Gamma(u)} h(v)\Big) = \sum_{(u,v)\in E} h(u)h(v) \\ &= \sum_{v\in V}\Big(h(v)\cdot\sum_{u\in\Gamma(v)} h(u)\Big) \ge B\sum_{v\in V} h(v). \end{aligned}$$

As $A > 0$ and $\sum_v h(v) \ge B > 0$, we conclude that

$$\frac{\sum_{u\in U} h(u)}{\sum_{v\in V} h(v)} \ge \frac{B}{A},$$

as desired. ☐

We now establish an upper bound on the competitive ratio of algorithm RS.

THEOREM 3.3. *Algorithm RS is $(1 + \mathbf{cw}(C)/e)$-competitive for any $(p,t)$-DAG $C = (V, E)$.*

*Proof.* Let $C$ be a $(p,t)$-DAG; recall that associated with $C$ are the two functions $h : V \to [t] \cup \{0\}$ and $g : V \to 2^{[p]} \setminus \{\emptyset\}$. For a subgraph $C' = (V', E')$ of $C$, we let $H(C') = \sum_{v\in V'} h(v)$. Recall that $P_C(v)$ and $S_C(v)$ denote the predecessor and successor graphs of $C$ at $v$. Then we say that a vertex $v \in V$ is *saturated* if $H(P_C(v)) \le t$; otherwise, $v$ is *unsaturated.* Note that if $v$ is saturated, then the group $g(v)$ must complete $h(v)$ tasks *regardless of the scheduling algorithm used.* Along these same lines, if $v$ is an unsaturated vertex for which $t > \sum_{u<v} h(u)$, the group $g(v)$ must complete at least $\max(h(v), t - \sum_{u<v} h(u))$ tasks under any scheduling algorithm. As these portions of $C$ which correspond to computation that must be performed by

any algorithm will play a special role in the analysis, it will be convenient for us to rearrange the DAG so that all such work appears on saturated vertices. To achieve this, note that if $v$ is an unsaturated vertex for which $\sum_{u<v} h(u) < t$, we may replace $v$ with a pair of vertices, $v_s$ and $v_u$, where all edges directed into $v$ are redirected to $v_s$, all edges directed out of $v$ are changed to originate at $v_u$, the edge $(v_s, v_u)$ is added to $E$, and $h$ is redefined so that

$$h(v_s) = t - \sum_{u<v} h(u) \qquad \text{and} \qquad h(v_u) = h(v) - h(v_s).$$

Note that the graph $C'$ obtained by altering $C$ in this way corresponds to the same computation, in the sense that $W_D(C) = W_D(C')$ for any algorithm $D$. For the remainder of the proof we will assume that this alteration has been made at every relevant vertex, so that the graph $C$ satisfies the condition

$$v \text{ unsaturated } \Rightarrow \sum_{u<v} h(u) \geq t. \tag{3.1}$$

Finally, for a vertex $v$, we let $T_v$ be the random variable equal to the number of tasks that RS completes at vertex $v$. Note that if $v$ is saturated, then $T_v = h(v)$. Let $\mathcal{S}$ and $\mathcal{U}$ denote the sets of saturated and unsaturated vertices, respectively. Given the above definitions, we immediately have

$$W_{\mathrm{OPT}} \geq \sum_{s\in\mathcal{S}} h(s)$$

and, by linearity of expectation,

$$W_{\mathrm{RS}} = \mathbb{E}\Big[\sum_v T_v\Big] = \sum_{s\in\mathcal{S}} h(s) + \sum_{u\in\mathcal{U}} \mathbb{E}[T_u] \leq W_{\mathrm{OPT}} + \sum_{u\in\mathcal{U}} \mathbb{E}[T_u]. \tag{3.2}$$

Our goal is to conclude that for some appropriate $\beta$,

$$\mathbb{E}\left[\sum_{u\in\mathcal{U}} T_u\right] \leq \beta \cdot \sum_{s\in\mathcal{S}} h(s) \leq \beta \cdot W_{\mathrm{OPT}}$$

and hence that RS is $1 + \beta$ competitive. We will obtain such a bound by applying Lemma 3.2 to an appropriate bipartite graph, constructed next.

Given $C = (V, E)$, construct the (undirected) bipartite graph $G = (\mathcal{S}, \mathcal{U}, E_G)$, where $E_G = \{(s, u) \mid s < u\}$. As in Lemma 3.2, for a vertex $v$, we let $\Gamma(v)$ denote the set of vertices adjacent to $v$. Now assign weights to the vertices of $G$ according to the rule $h^*(v) = \mathbb{E}[T_v]$. Note that for $s \in \mathcal{S}, h^*(s) = h(s)$ and hence by condition (3.1) above, we immediately have the bound

$$\forall u \in \mathcal{U}, \quad \sum_{s\in\Gamma(u)} h^*(s) \geq t. \tag{3.3}$$

We now show that $\forall s \in \mathcal{S}$,

$$\sum_{u\in\Gamma(s)} h^*(u) \leq \mathbf{cw}(C) \cdot \frac{t}{e}. \tag{3.4}$$

Before proceeding to establish this bound, note that (3.3) and (3.4), together with Lemma 3.2, imply that

$$\begin{aligned} W_{\mathrm{RS}}(C) &\leq \sum_{s\in\mathcal{S}} h(s) + \sum_{u\in\mathcal{U}} h^*(u) \leq \Big(1+\frac{\mathbf{cw}(C)}{e}\Big)\sum_{s\in\mathcal{S}} h(s) \\ &\leq \Big(1+\frac{\mathbf{cw}(C)}{e}\Big) W_{\mathrm{OPT}}(C), \end{aligned}$$

as desired.

Returning now to (3.4), let $s \in \mathcal{S}$ be a saturated vertex and consider the successor graph (of $C$) at $s$, $S_C(s)$. By Lemma 3.1 (Dilworth's lemma), there exist $w \triangleq \mathbf{w}(S_C(s)) \leq \mathbf{cw}(C)$ paths in $S_C(s)$, $P_1, P_2, \ldots, P_w$, so that their union covers $S_C(s)$. Let $X_i$ be the random variable whose value is the number of tasks performed by RS on the portion of the path $P_i$ consisting of unsaturated vertices. Note that if $u \in V$ is unsaturated and $u \leq v$, then $v$ is unsaturated and hence, for each path $P_i$, there is a first unsaturated vertex $u_i^0$ after which every vertex of $P_i$ is unsaturated. Note now that for a fixed individual task $\tau$, conditioned upon the event that $\tau$ is not yet complete, the probability that $\tau$ is *not* chosen by RS for completion at a given selection point in $P_C(u_i^0)$ is no more than $(1-1/t)$. Let $L_i$ be the random variable whose value is the set of tasks left incomplete by RS at the formation of the group $g(u_i^0)$. As $u_i^0$ is unsaturated, $\sum_{v<u_i^0} h(v) \geq t$ by condition (3.1) and hence, for each $i$,

$$\Pr[\tau \in L_i] \leq (1-1/t)^t \leq 1/e.$$

As there are a total of $t$ tasks,

$$\mathbb{E}[|L_i|] \leq t/e.$$

Of course, since RS completes a new task at each step, $X_i \leq |L_i|$ so that $\mathbb{E}[X_i] \leq t/e$, and by linearity of expectation

$$\mathbb{E}\Big[\sum_i X_i\Big] \leq w \cdot t/e.$$

Now every unsaturated vertex in $S_C(s)$ appears in some $P_i$ and hence

$$\sum_{u\in\Gamma(s)} h^*(u) \leq \mathbb{E}\Big[\sum_i X_i\Big] \leq wt/e \leq \mathbf{cw}(C)\cdot t/e,$$

as desired. ◻

Theorem 3.3 implies a constant upper bound for patterns that consist entirely of merges (that is, where all reconfigurations are given by taking unions of existing groups). This subsumes the results reported in [14].

COROLLARY 3.4. *Algorithm RS is $\big(1+\frac{1}{e}\big)$-competitive for any $(p,t)$-DAG $C$ with* $\mathbf{cw}(C) = 1$.

*Remark.* The proof of Theorem 3.3 can be slightly modified to yield an interesting result for *deterministic* scheduling algorithms. Let $D$ be a deterministic scheduling algorithm for *Omni-Do*. In the proof of Theorem 3.3, $h^*(v)$ was defined as the expected number of tasks performed by algorithm RS at node $v$. For algorithm $D$, if we define $h^*(v)$ to be the actual number of tasks performed by the algorithm at node $v$, then it is not difficult to see that (3.4) becomes $\sum_{u\in\Gamma(s)} h^*(u) \leq \mathbf{cw}(C)\cdot t$ (provided that no processor in $D$ performs a task that already knows its result). This leads to the conclusion that *any (nontrivial) deterministic algorithm for Omni-Do is* $(1+\mathbf{cw}(C))$*-competitive for any computation pattern $C$.*

**4. A lower bound.** We begin with a lower bound for *deterministic* algorithms. This is then applied to give a lower bound for randomized algorithms in Corollary 4.2.

THEOREM 4.1. *Let $a : \mathbb{N} \to \mathbb{R}$ and $D$ be a deterministic scheduling algorithm for Omni-Do so that $D$ is $a(\mathbf{cw}(\cdot))$-competitive (that is, $D$ is $\alpha$-competitive, for a function $\alpha = a \circ \mathbf{cw}$)). Then $a(c) \geq 1 + c/e$.*

*Proof.* Fix $k \in \mathbb{N}$. Consider the case when $t = p = g \gg k$ and $t \bmod k = 0$, $g$ being the number of initial groups. We consider a computation pattern $C_{\mathbf{G}}$ determined by a tuple $\mathbf{G} = (G_1, \dots, G_{t/k})$, where each $G_i \subset [t]$ is a set of size $k$ and $\bigcup_i G_i = [t]$. Initially, the computation pattern $C_{\mathbf{G}}$ has the processors synchronously proceed until each has completed $t/k$ tasks; at this point, the processors in $G_i$ are merged and allowed to exchange information about task executions. Each $G_i$ is then immediately partitioned into $c$ groups (this establishes that the computation width is $c$). Note that the off-line optimal algorithm accrues exactly $t^2/k$ work for this computation history (it terminates prior to the partitions of the $G_i$).

We will show that for *any* scheduling deterministic Omni-Do algorithm $D$, there is a selection of the $G_i$ so that

$$W_D(C_{\mathbf{G}}) \geq t^2/k \left[1 + c\left(1 - \frac{1}{k}\right)^k - o(1)\right],$$

and hence that $a(c) \geq 1 + c/e$. Consider the behavior of $D$ when $\mathbf{G}$ is selected at random, uniformly among all such tuples. Let $P_i \subset [t]$ be the subset of $t/k$ tasks completed by processor $i$ before the merges take place; these sets are determined by the algorithm $D$. We begin by bounding

$$\mathbb{E}_{\mathbf{G}}\left[\left|\bigcup_{i \in G_1} P_i\right|\right].$$

To this end, consider an experiment where we select $k$ sets $Q_1, \dots, Q_k$, each $Q_i$ selected independently and uniformly from the set $\{P_i\}$. Now, for a specific task $\tau$, let $p_\tau = \Pr_{Q_1}[\tau \notin Q_1]$, so that $\Pr_{Q_i}[\tau \notin \bigcup_i Q_i] = p_\tau^k$. As the $Q_i$ are selected independently,

$$\mathbb{E}_{Q_i}\left[\left|[t] - \bigcup_i Q_i\right|\right] = \sum_\tau p_\tau^k.$$

Observe now that

$$\sum_\tau (1 - p_\tau) = \sum_\tau \Pr_{Q_1}[\tau \in Q_1] = \mathbb{E}_{Q_1}[|Q_1|] = t/k$$

and hence $\sum_\tau p_\tau = t(1 - 1/k)$. As the function $x \mapsto x^k$ is convex on $[0, \infty)$, $\sum_\tau p_\tau^k$ is minimized when the $p_\tau$ are equal, and we must have

$$\mathbb{E}_{Q_i}\left[\left|[t] - \bigcup_i Q_i\right|\right] \geq t \cdot \left(1 - \frac{1}{k}\right)^k.$$

Now observe that, conditioned on the $Q_i$ being distinct, the distribution of $(Q_1, \dots, Q_k)$ is identical to that of $(P_{g_1^1}, \dots, P_{g_k^1})$, where the random variable $G_1 = \{g_1^1, \dots, g_k^1\}$. Considering that $\Pr[\exists i \neq j, Q_i = Q_j] \leq k^2/t$, we have

$$\mathbb{E}_{Q_i}\left[\left|[t] - \bigcup_i Q_i\right|\right] \leq \left(1 - \frac{k^2}{t}\right) \mathbb{E}_{\mathbf{G}}\left[t - \left|\bigcup_{i \in G_1} P_i\right|\right] + 1 \cdot \frac{k^2}{t},$$

and hence as $t \to \infty$, we see that the expected number of tasks remaining for those processors in group $G_1$ is

$$\mathbb{E}_{\mathbf{G}}\left[t - \Big|\bigcup_{i\in G_1} P_i\Big|\right] \geq t(1-1/k)^k - o(1).$$

Of course, the distribution of each $G_i$ is the same, so that

$$\mathbb{E}_{\mathbf{G}}\left[\sum_{i=1}^{t/k}\left(t - \Big|\bigcup_{j\in G_i} P_j\Big|\right)\right] = [1-o(1)]\left(\frac{t}{k}\right)\cdot t\left(1-\frac{1}{k}\right)^k.$$

In particular, there must exist a specific selection of $\mathbf{G} = (G_1, \dots, G_{t/k})$ which achieves this bound. Recall that every $G_i$ is partitioned into $c$ groups. Therefore, for such $\mathbf{G}$, the total work is at least

$$\frac{t^2}{k}\cdot\left(1 + [1-o(1)]\cdot c\cdot\left(1-\frac{1}{k}\right)^k\right).$$

As $\lim_{k\to\infty}(1-\frac{1}{k})^k = \frac{1}{e}$, this completes the proof. ∎

As the above stochastic computation pattern $C_{\mathbf{G}}$ is independent of the deterministic algorithm $D$, this immediately gives rise to a lower bound for randomized algorithms.

COROLLARY 4.2. *Let $\mathcal{R}\big(\{D_r \mid r\in\Omega\}\big)$ be a randomized scheduling algorithm for Omni-Do that is $(a\circ\mathbf{cw})$-competitive. Then $a(c) \geq 1 + c/e$.*

*Proof.* Assume for contradiction that for some $c$, $a(c) < 1 + c/e$, and let $k$ be large enough so that $(1-\frac{1}{k})^k > a(c) - 1$. For this $k$ we proceed as in the proof above, considering a random $\mathbf{G}$ and the computation pattern $C_{\mathbf{G}}$ with $t = g = p$ congruent to 0 mod $k$, $g$ being the number of initial groups. Then, as above,

$$\begin{aligned}\mathbb{E}_{\mathbf{G}}\left[\mathbb{E}_r\left[W_{D_r}(C_{\mathbf{G}})\right]\right] &= \mathbb{E}_r\left[\mathbb{E}_{\mathbf{G}}\left[W_{D_r}(C_{\mathbf{G}})\right]\right]\\ &\geq \min_r\left[\mathbb{E}_{\mathbf{G}}\left[W_{D_r}(C_{\mathbf{G}})\right]\right]\\ &\geq \frac{t^2}{k}\cdot\left(1 + [1-o(1)]\cdot c\cdot\left(1-\frac{1}{k}\right)^k\right).\end{aligned}$$

Hence there exists a $\mathbf{G}$ so that $\mathbb{E}_r\left[W_{D_r}(C_{\mathbf{G}})\right] \geq \frac{t^2}{k}\cdot\left(1 + [1-o(1)]\frac{c}{e}\right)$, which completes the proof. ∎

**5. Conclusions and open problems.** We established bounds on the competitive ratio of a natural randomized algorithm for scheduling in partitionable networks and show, furthermore, that for the relevant gradation of computation patterns these bounds are tight. We showed how to characterize algorithm competitiveness in terms of computation width, a precise property of a DAG that describes the computation history. These results lead to a better understanding of the effectiveness of computation in *group communication schemes*, a widely used paradigm for computing in distributed environments.

One outstanding open question is how to derandomize the schedules used by task-performing algorithms in this work. Specifically, we would like to construct deterministic scheduling algorithms that are $(1 + \mathbf{cw}(C)/e)$-competitive for any computation pattern $C$. Another promising direction is to study the task-performing paradigm in the models of computation that combine network reconfigurations with processor failures. The goal is to establish complexity results that show how performance of task-performing algorithms depends both on the extent of the network reconfiguration and on the number of processor failures.

## REFERENCES

[1] M. Ajtai, J. Aspnes, C. Dwork, and O. Waarts, *A theory of competitive analysis for distributed algorithms*, in Proceedings of the 35th Symposium on Foundations of Computer Science (FOCS 1994), IEEE, Los Alamitos, CA, 1994, pp. 401–411.

[2] R. J. Anderson and H. Woll, *Algorithms for the certified write-all problem*, SIAM J. Comput., 26 (1997), pp. 1277–1283.

[3] B. Awerbuch, S. Kutten, and D. Peleg, *Competitive distributed job scheduling*, in Proceedings of the 24th ACM Symposium on Theory of Computing (STOC 1992), ACM, New York, 1992, pp. 571–580.

[4] O. Babaoglu, R. Davoli, A. Montresor, and R. Segala, *System support for partition-aware network applications*, in Proceedings of the 18th IEEE International Conference on Distributed Computing Systems (ICDCS 1998), IEEE, Los Alamitos, CA, 1998, pp. 184–191.

[5] Y. Bartal, A. Fiat, and Y. Rabani, *Competitive algorithms for distributed data management*, in Proceedings of the 24th ACM Symposium on Theory of Computing (STOC 1992), ACM, New York, 1992, pp. 39–50.

[6] S. Ben-David, A. Borodin, R. Karp, G. Tardos, and A. Wigderson, *On the power of randomization in on-line algorithms*, Algorithmica, 11 (1994), pp. 2–14.

[7] B. Chlebus, R. De Prisco, and A. A. Shvartsman, *Performing tasks on restartable message-passing processors*, Distributed Comput., 14 (2001), pp. 49–64.

[8] R. De Prisco, A. Mayer, and M. Yung, *Time-optimal message-efficient work performance in the presence of faults*, in Proceedings of the 13th ACM Symposium on Principles of Distributed Computing (PODC 1994), ACM, New York, 1994, pp. 161–172.

[9] R. P. Dilworth, *A decomposition theorem for partially ordered sets*, Ann. of Math., 51 (1950), pp. 161–166.

[10] S. Dolev, R. Segala, and A. A. Shvartsman, *Dynamic load balancing with group communication*, in Proceedings of the 6th International Colloquium on Structural Information and Communication Complexity (SIROCCO 1999), Carleton Scientific, Waterloo, ON, Canada, 1999, pp. 111–125.

[11] C. Dwork, J. Y. Halpern, and O. Waarts, *Performing work efficiently in the presence of faults*, SIAM J. Comput., 27 (1998), pp. 1457–1491.

[12] A. Fiat, R. M. Karp, M. Luby, L. A. McGeoch, D. D. Sleator, and N. E. Young, *Competitive paging algorithms*, J. Algorithms, 12 (1991), pp. 685–699.

[13] S. Georgiades, M. Mavronicolas, and P. Spirakis, *Optimal, distributed decision-making: The case of no communication*, in Proceedings of the 12th International Symposium on Fundamentals of Computation Theory (FCT 1999), Lecture Notes in Comput. Sci. 1684, Springer-Verlag, Berlin, 1999, pp. 293–303.

[14] Ch. Georgiou, A. Russell, and A. A. Shvartsman, *Optimally work-competitive scheduling for cooperative computing with merging groups (brief announc.)*, in Proceedings of the 22nd ACM Symposium on Principles of Distributed Computing (PODC 2002), ACM, New York, 2002, p. 132.

[15] Ch. Georgiou, A. Russell, and A. A. Shvartsman, *Work-competitive scheduling for cooperative computing with dynamic groups*, in Proceedings of the 35th Annual ACM Symposium on Theory of Computing (STOC 2003), ACM, New York, 2003, pp. 251–258.

[16] Ch. Georgiou and A. A. Shvartsman, *Cooperative computing with fragmentable and mergeable groups*, J. Discrete Algorithms, 1 (2003), pp. 211–235.

[17] J. F. Groote, W. H. Hesselink, S. Mauw, and R. Vermeulen, *An algorithm for the asynchronous Write-All problem based on process collision*, Distributed Comput., 14 (2001), pp. 75–81.

[18] P. C. Kanellakis and A. A. Shvartsman, *Fault-Tolerant Parallel Computation*, Kluwer Academic, Dordrecht, The Netherlands, 1997.

[19] Z. M. Kedem, K. V. Palem, and P. Spirakis, *Efficient robust parallel computations*, in Proceedings of the 22nd ACM Symposium on Theory of Computing (STOC 1990), ACM, New

York, 1990, pp. 138–148.

[20] G. Malewicz, A. Russell, and A. A. Shvartsman, *Distributed cooperation during the absence of communication*, in Proceedings of the 14th International Symposium on Distributed Computing (DISC 2000), Lecture Notes in Comput. Sci. 1914, Springer-Verlag, Berlin, 2000, pp. 119–133.

[21] C. Martel and R. Subramonian, *On the complexity of certified Write-All algorithms*, J. Algorithms, 16 (1994), pp. 361–387.

[22] C. H. Papadimitriou and M. Yannakakis, *On the value of information in distributed decision-making*, in Proceedings of the 10th ACM Symposium on Principles of Distributed Computing (PODC 1991), ACM, New York, 1991, pp. 61–64.

[23] D. Powell, ed., *Special Issue on Group Communication Services*, Comm. ACM, 39 (1996).

[24] M. Saks, N. Shavit, and H. Woll, *Optimal time randomized consensus—making resilient algorithms fast in practice*, in Proceedings of the 2nd ACM-SIAM Symposium on Discrete Algorithms (SODA 1991), ACM, New York, 1991, pp. 351–362.

[25] N. Shavit, *Concurrent Timestamping*, Ph.D. thesis, The Hebrew University, 1989.

[26] D. Sleator and R. Tarjan, *Amortized efficiency of list update and paging rules*, Comm. ACM, 28 (1985), pp. 202–208.

[27] J. B. Sussman and K. Marzullo, *The Bancomat problem: An example of resource allocation in a partitionable asynchronous system*, in Proceedings of the 12th International Symposium on Distributed Computing (DISC 1998), Lecture Notes in Comput. Sci. 1499, Springer-Verlag, Berlin, 1998, pp. 363–377.

SIAM J. Comput.
Vol. 34, No. 4, pp. 863–878

# CURVE-SENSITIVE CUTTINGS*

VLADLEN KOLTUN† AND MICHA SHARIR‡

**Abstract.** We introduce $(1/r)$-cuttings for collections of surfaces in 3-space, such that the cuttings are sensitive to an additional collection of curves. Specifically, let $S$ be a set of $n$ surfaces and let $C$ be a set of $m$ curves in $\mathbb{R}^3$, all of constant description complexity. Let $1 \le r \le \min\{m, n\}$ be a given parameter. We show the existence of a $(1/r)$-cutting $\Xi$ of $S$ of size $O(r^{3+\varepsilon})$, for any $\varepsilon > 0$, such that the number of crossings between the curves of $C$ and the cells of $\Xi$ is $O(mr^{1+\varepsilon})$. The latter bound improves, by roughly a factor of $r$, the bound that can be obtained for cuttings based on vertical decompositions. We view curve-sensitive cuttings as a powerful tool for various scenarios that involve curves and surfaces in three dimensions. As a preliminary application, we use the construction to obtain a bound of $O(m^{1/2}n^{2+\varepsilon})$, for any $\varepsilon > 0$, on the complexity of the multiple zone of $m$ curves in the arrangement of $n$ surfaces in 3-space. After the conference publication of this paper [V. Koltun and M. Sharir, *Proceedings of the* 19*th ACM Symposium on Computational Geometry*, 2003, pp. 136–143], curve-sensitive cuttings were applied to derive an algorithm for efficiently counting triple intersections among planar convex objects in three dimensions [E. Ezra and M. Sharir, *Proceedings of the* 20*th ACM Symposium on Computational Geometry*, 2004, pp. 210–219], and we expect additional applications to arise in the future.

**Key words.** computational geometry, cuttings, random sampling, curves in space, zone

**AMS subject classifications.** 68U05, 52C45, 68W20

**DOI.** 10.1137/S0097539703435686

## 1. Introduction.

**Motivation.** $(1/r)$-cuttings (see below for definitions) have attracted considerable attention in the computational geometry community, as they turned out to be crucial to the solution of many central problems in the field [5, 6, 7, 8, 9, 10, 14, 16, 17]. For some applications, special properties possessed by the cutting can lead to improved results. For instance, the tree structure of *hierarchical cuttings* [6] is of great help in numerous settings [4, 17].

We construct a $(1/r)$-cutting for a collection of surfaces in 3-space, such that the cutting is sensitive, in the sense defined below, to a collection of curves given as additional input to the construction. We apply this cutting to obtain a bound of $O(m^{1/2}n^{2+\varepsilon})$, for any $\varepsilon > 0$, on the complexity of the multiple zone of $m$ curves in the arrangement of $n$ surfaces in 3-space, all of constant description complexity. The multiple zone is defined as the collection of all cells of the arrangement of the given surfaces that are crossed by at least one of the curves. It is a generalization of both the concept of the zone of a curve in an arrangement [3, 13] and the widely studied notion of many faces/cells in arrangements [2].

*Received by the editors October 6, 2003; accepted for publication (in revised form) December 17, 2004; published electronically May 12, 2005. Work on this paper was initiated at the DIMACS Workshop on Geometric Graph Theory, held at Rutgers University, Fall 2002. This work has been supported by a grant from the U.S.-Israeli Binational Science Foundation, by a grant from the Israel Science Fund (for the Center of Excellence in Geometric Computing), by NSF grants CCR-97-32101, CCR-00-98246, and CCR-01-21555, and by the Hermann Minkowski–MINERVA Center for Geometry at Tel Aviv University.

http://www.siam.org/journals/sicomp/34-4/43568.html

†Computer Science Division, University of California, Berkeley, CA 94720-1776 (vladlen@cs.berkeley.edu).

‡School of Computer Science, Tel Aviv University, Tel-Aviv 69978, Israel, and Courant Institute of Mathematical Sciences, New York University, New York, NY 10012 (michas@post.tau.ac.il).

We expect curve-sensitive cuttings to find additional uses in contexts that involve the interaction of curves and surfaces. It has already been applied, after the conference publication of this paper [15], to derive an algorithm for efficiently counting triple intersections among planar convex objects in three dimensions [12].

**Overview.** Let $S$ be a set of $n$ surfaces in $\mathbb{R}^3$ of constant description complexity, and let $C$ be a set of $m$ curves in $\mathbb{R}^3$ of constant description complexity; that is, each surface and curve is defined as a Boolean combination of a constant number of polynomial equations and inequalities of constant maximum degree. Let $1 \leq r \leq \min\{m, n\}$ be a given parameter. A $(1/r)$-*cutting* of $S$ is a subdivision of 3-space into connected cells, each of constant description complexity, so that each cell is crossed by at most $n/r$ surfaces of $S$. We wish to construct a $(1/r)$-cutting $\Xi$ of $S$ of size near $O(r^3)$, so that the number of pairs $(c, \tau)$, where $c \in C$, $\tau$ is a cell of $\Xi$, and $c \cap \tau \neq \emptyset$, is near $O(mr)$; that is, the average number of cells of $\Xi$ crossed by a curve of $C$ is near $O(r)$.

A standard method (in fact, the only general-purpose method known to date) for constructing a $(1/r)$-cutting for arrangements of nonlinear surfaces is to take an appropriate random sample $R$ of the surfaces of $S$ and construct the *vertical decomposition* of the arrangement $\mathcal{A}(R)$ of $R$ [18]. The construction of this decomposition proceeds in two stages. First, for every edge of $\mathcal{A}(R)$ and every vertical tangency curve (also known as the *silhouette*) on every surface of $R$, we erect a two-dimensional vertical *visibility wall*, defined as the union of all $z$-vertical segments that have an endpoint on this edge (or curve) and are interior-disjoint from all surfaces of $R$. This first stage results in a decomposition of $\mathcal{A}(R)$ into vertical pseudoprisms, such that the floor of each prism, if it exists, is contained in a single surface of $R$, and similarly for the ceiling of each prism. However, the combinatorial complexity of a single prism can still be fairly high.

In the second stage of the construction we refine the decomposition as follows. For every prism as above, consider its projection onto the $xy$-plane. It is a two-dimensional semialgebraic set, which we decompose in the plane by erecting zero, one, or two $y$-vertical (possibly infinite) visibility segments on each of its vertices and $y$-vertical tangency points on its edges, where a visibility segment is defined as a maximal $y$-vertical segment that has an endpoint on this vertex (or tangency point), is contained in the considered prism projection, and is interior-disjoint from its boundary. We then erect $z$-vertical two-dimensional walls inside the original prism, defined as its intersection with the $z$-vertical walls spanned by all the $y$-vertical segments erected by the planar decomposition. Repeating this process for each of the above prisms decomposes $\mathcal{A}(R)$ into cells of constant description complexity.

We can choose $R$ as a single sample from $S$ of size $ar \log r$, for an appropriate absolute constant $a$. It can then be argued that, with high probability, the resulting vertical decomposition of $\mathcal{A}(R)$ is indeed a $(1/r)$-cutting. This is a consequence of the probabilistic analyses of Haussler and Welzl [14] and of Clarkson [9]. Using a variant of the method of Chazelle and Friedman [7] or of Chazelle [6] slightly reduces the size of the resulting cutting from $O(r^3 \log^3 r)$ to $O(r^3)$.

Unfortunately, vertical decompositions may fail to satisfy our requirement concerning the number of crossings between the curves of $C$ and the cells of the cutting. In fact, a curve may cross nearly $\Omega(r^2)$ such cells. An example is shown in Figure 1, where $R$ is a collection of $r$ planes. Half of them are parallel to the $x$-axis and pass above it, all appearing on the lower envelope of this group, which looks like a tunnel in the $x$-direction with a convex roof that is symmetric about the $xz$-plane. The remain-

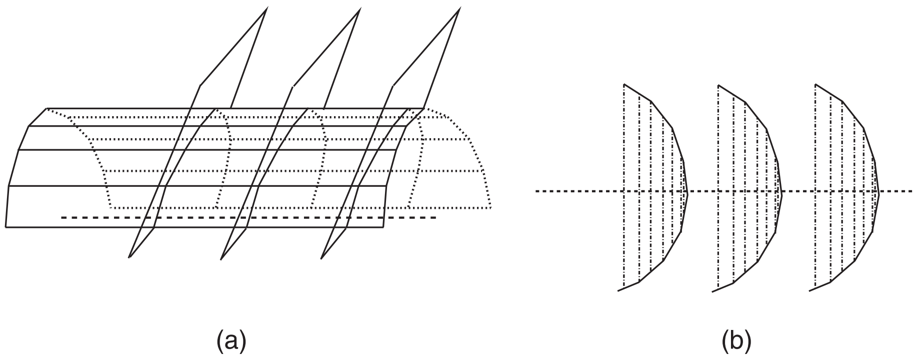


FIG. 1. *A curve (the x-axis, shown dashed) crossing a quadratic number of cells of the vertical decomposition.* (a) *A side view of the input set.* (b) *A view from above of the second-step subdivision of the cells mentioned in the text.*

ing $r/2$ planes are all parallel to the $y$-axis and form a fixed angle, say $45°$, with the $xy$-plane. These latter planes are sufficiently separated from each other so that their portions that lie above the $xy$-plane and below the lower envelope of the first group have pairwise disjoint $xy$-projections. The $x$-axis crosses $\Theta(r^2)$ cells of the vertical decomposition of these planes: Indeed, the first decomposition step creates (among others) $r/2$ cells whose top facet is the portion of some slanted plane of the second group that lies below the lower envelope of the first group. The second decomposition step subdivides each of these cells into $\Theta(r)$ subcells, and the $x$-axis crosses them all.

In contrast, the *undecomposed* arrangement of $\Theta(r \log r)$ surfaces is sensitive to the curves of $C$, because each curve crosses each surface at $O(1)$ points, so it crosses $O(r \log r)$ cells of the arrangement. However, the undecomposed arrangement is generally not a $(1/r)$-cutting. On the other hand, the decomposed arrangement is (with high probability) a $(1/r)$-cutting, but, as we have just seen, it may fail to be sensitive to $C$. (Actually, as we will show in section 2.1, the first stage of the vertical decomposition is also sensitive to $C$, but in general it is still not a $(1/r)$-cutting.)

In this paper we describe a technique that achieves the better of both worlds and constructs cuttings that satisfy the desired properties. The construction proceeds by taking a sample $R$ of the surfaces, as described above, and decomposing $\mathcal{A}(R)$ into vertical prisms using the *first stage* of the vertical decomposition construction. Inside each prism we construct a decomposition that takes into account the parts of the curves of $C$ that lie inside the prism. Specifically, we construct a hierarchical sequence of cuttings, somewhat reminiscent of the construction in Chazelle [6], that reduces the number of crossings between the curves of $C$ and the boundaries of the cells of the cuttings. We are able to guarantee that the curves of $C$ are not cut more than $O(mr^{1+\varepsilon})$ times, for any $\varepsilon > 0$, overall.

Before describing our results in detail, we remark that we can construct an alternative (and simpler) curve-sensitive decomposition scheme for the special case where the surfaces are planes and the curves are lines (as in the example of Figure 1) by using the Dobkin–Kirkpatrick hierarchical decomposition [11] in each cell of $\mathcal{A}(R)$. This approach, however, does not extend to general curves and surfaces. (An expanded discussion of this remark is given in the application paper [12].)

**2. A curve-sensitive decomposition.** In this section we present a new decomposition scheme that is a $(1/r)$-cutting for $S$ and satisfies the desired bounds on the number of cells and on the number of curve-cell crossings. For simplicity of exposition, we will base our analysis on a single random sample of surfaces from $S$ (rather than the more elaborate repeated-sampling scheme of [7]). Moreover, we consider samples of size $r$ (rather than $\Theta(r \log r)$). This simplifies the calculations, but will only produce an $O(\log r/r)$-cutting. We get the desired cutting by simply replacing $r$, at the end of the analysis, by the above larger sample size.

**2.1. First stage of the decomposition.** We begin with taking a random sample $R$ of $r$ surfaces of $S$, and a random sample $R'$ of $r$ curves of $C$. We form the arrangement $\mathcal{A}(R)$ of $R$ and apply to it the first step of the vertical decomposition. That is, we erect vertical walls up and down from each curve of an intersection of pairs of surfaces in $R$, as well as from the silhouette of each surface in $R$; the walls are extended until they hit another surface of $R$ or, failing that, all the way to $\pm\infty$. In addition, we erect similar vertical walls from each curve $c \in R'$, which are also extended to the first surface above and below.

Let $\mathcal{A}_1 = \mathcal{A}_1(R, R')$ denote the resulting decomposition. Note that each cell $\tau$ of $\mathcal{A}_1$ is a vertical prism-like cell: the intersection of each vertical line with $\tau$ is connected. However, the $xy$-projection $\tau^*$ of $\tau$ can have arbitrary shape and complexity.

For each cell $\tau$ of $\mathcal{A}_1$, let $\xi_\tau$ denote its combinatorial complexity (i.e., the number of vertices, edges, and faces on its boundary), and let $C_\tau$ denote the set of all connected components of the nonempty intersections between $\tau$ and the curves of $C$. Let $\lambda_q(r)$ denote, as usual, the maximum length of a Davenport–Schinzel sequence of order $q$ on $r$ symbols [18], and put $\beta_q(r) = \lambda_q(r)/r$, which is thus an extremely slow-growing function of $r$. We have the following lemma.

LEMMA 2.1. (a) *The number of cells of $\mathcal{A}_1$ and their overall combinatorial complexity are both $O(r^3\beta_q(r))$ for an appropriate parameter $q$ that depends on the algebraic complexity of the curves of $C$ and the surfaces of $S$.*

(b) $\sum_{\tau\in\mathcal{A}_1} |C_\tau| = O(mr\beta_q(r))$.

*Proof.* Let $\gamma$ be a fixed curve, which is either a curve in $C$, an intersection curve of two surfaces in $R$, or the silhouette of a surface in $R$. Let $V_\gamma$ denote the vertical 2-manifold (wall) spanned by $\gamma$. Let $V_\gamma^+$ (resp., $V_\gamma^-$) denote the portion of $V_\gamma$ that lies above (resp., below) $\gamma$. Let $\mathcal{A}^+$ (resp., $\mathcal{A}^-$) denote the cross section of $\mathcal{A}(R)$ with $V_\gamma^+$ (resp., $V_\gamma^-$). By construction, any point at which $\gamma$ crosses the boundary of some cell of $\mathcal{A}_1$ must be either the vertical projection on $\gamma$ of a vertex of the lower envelope of $\mathcal{A}^+$, a vertex of the upper envelope of $\mathcal{A}^-$ (or of both, if the vertex lies on $\gamma$ itself), or a point that lies vertically above or below a point on another curve of $R'$ (so that the two points are *vertically visible* in $\mathcal{A}(R)$). The complexity of each envelope is $O(\lambda_q(r)) = O(r\beta_q(r))$ for an appropriate constant $q$ [18], and the number of times $\gamma$ passes above or below any curve of $R'$ is $O(r)$ (over all curves of $R'$). This readily implies the lemma: Part (b) is an immediate consequence, while part (a) follows by applying this bound to each of the $O(r^2)$ intersection and $O(r)$ silhouette curves arising in the sample. □

**2.2. Second stage of the decomposition.** After constructing the decomposition $\mathcal{A}_1$, we perform a second decomposition step, which decomposes each cell $\tau$ of $\mathcal{A}_1$ as follows. Let $\partial\tau^*$ denote the boundary of $\tau^*$ and let $h_\tau$ denote the number of internal boundary components ("holes") of $\tau^*$. Note that $h_\tau \le \xi_\tau$. Since $\tau^*$ need not be simply connected, $\partial\tau^*$ may consist of more than one connected component (i.e., $h_\tau$ may be strictly positive). The potential existence of many components of $\partial\tau^*$ is the main source of technical difficulty in the analysis of our decomposition.

Put $m_\tau = |C_\tau|$. Let $C^*_\tau$ denote the set of the $xy$-projections of the arcs of $C_\tau$. Let $X_\tau$ denote the number of intersections between the curves of $C^*_\tau$. This is also equal to the number of *vertical visibility segments* between pairs of curves of $C_\tau$, where such a segment is parallel to the $z$-axis and connects a point on one curve to a point on the other (and is thus fully contained in $\tau$). We clearly have

$$\sum_{\tau \in \mathcal{A}_1} X_\tau = O(m^2). \tag{1}$$

In what follows, through the bulk of this section, we assume that $X_\tau \geq m_\tau$. The alternative case $X_\tau < m_\tau$ is considerably simpler to handle and will be described later in the analysis.

If $X_\tau = o(m_\tau^2)$, we carry out a preliminary decomposition stage that covers $\tau^*$ by the union of simpler-shaped subcells, so that, within each such subcell $\tau_0$, the number of intersections between the curves of $C^*_\tau$ that cross $\tau_0$ is roughly the square of the number of such curves. We employ a standard approach that proceeds as follows. (See, e.g., [5].) Put $s = s_\tau = \lceil m_\tau^2 / X_\tau \rceil$. We distinguish between the following two cases.

(a) Suppose first that $s \leq \xi_\tau$. We sample each curve of $C^*_\tau$ with probability $s/m_\tau$. This produces a random sample $R''$ of expected size $s$. The expected complexity of $\mathcal{A}(R'')$ is $O(s + (s/m_\tau)^2 X_\tau) = O(s)$, since each intersection counted in $X_\tau$ becomes a vertex of $\mathcal{A}(R'')$ with probability $(s/m_\tau)^2$. We construct the vertical decomposition of $\mathcal{A}(R'')$ and argue that, with high probability, it consists of $O(s)$ trapezoids, each of which is crossed by at most $O((m_\tau/s)\log s)$ curves of $C^*_\tau$. We next apply a modified version of the analysis of Chazelle and Friedman [7] to refine the decomposition, so that each of its cells is crossed by at most $m_\tau/s$ curves of $C^*_\tau$, while the number of cells remains $O(s)$.

Since the setup here is somewhat different from that in [7], we present details of the construction and of its analysis. This is done as follows. We take each cell $\Delta$ of the vertical decomposition that is crossed by $tm_\tau/s$ curves of $C^*_\tau$, for any $t > 1$, draw a random sample $R''_\Delta$ of $ct\log t$ of these curves, for an appropriate sufficiently large constant $c$, construct the vertical decomposition of the arrangement $\mathcal{A}(R''_\Delta)$, and clip each resulting cell to $\Delta$. With high probability, each cell in the resulting decomposition is crossed by at most $m_\tau/s$ curves of $C^*_\tau$, provided $c$ is chosen sufficiently large. To estimate the overall number of cells, we apply Lemma 2.2 of Agarwal, Matoušek, and Schwarzkopf [1], which, in our context, asserts that the expected number of cells that are crossed by at least $jm_\tau/s$ curves is $O(2^{-j})$ times the expected number of cells in a random sample of $s/j$ curves of $C^*_\tau$. The latter expected number of cells is easily seen to be $O(s/j)$, and thus the overall expected number of new cells is

$$O(s) + \sum_{j\geq 1} O(2^{-j}s/j) = O(s),$$

as claimed.

As we do throughout the analysis, we assume that all those subsamples meet their expected values, so that this property holds with certainty. This assumption can be made effective, e.g., by resampling at each stage of the construction until a good sample is obtained. See a remark to that effect following Theorem 2.3.

These trapezoids are the cells $\tau_0$ of the cutting decomposition (or, rather, covering) of $\tau^*$. Each cell $\tau_0$ contains on average $X_\tau/s = O(X_\tau^2/m_\tau^2)$ crossings between curves of $C^*_\tau$, which is roughly the square of the number $O(m_\tau/s) = O(X_\tau/m_\tau)$ of

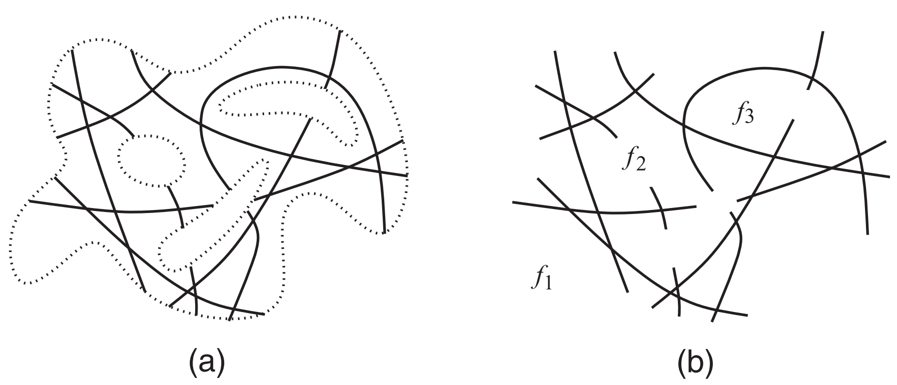


FIG. 2. *Stage* 2 *of the decomposition.* (a) *The curves of $Q \subseteq C^*_{\tau_0}$ (solid) and $\partial\tau^*$ (dotted).* (b) *The external faces of $\mathcal{A}(Q)$; note that $f_2$ contains two components of $\partial\tau^*$.*

these curves that cross $\tau_0$. It is important to note that this decomposition is defined only in terms of the curves in $C^*_\tau$, and is thus not necessarily confined within $\tau^*$. Thus our trapezoids constitute a *covering* of $\tau^*$. (Nevertheless, since all the curves of $C^*_\tau$ are fully contained in $\tau^*$, the portion of the covering outside $\tau^*$ is uninteresting; it is constructed simply because we do not want at this stage to let $\partial\tau^*$ affect the construction.) We shall later, towards the end of this section, take care to clip the new cells to within $\tau^*$.

(b) Suppose next that $s > \xi_\tau$. We then sample each curve of $C^*_\tau$ with probability $\xi_\tau/m_\tau$. Note that this quantity is indeed at most 1, because $s \le m_\tau$ (which follows from the assumption $X_\tau \ge m_\tau$) and $\xi_\tau < s$. This produces a random sample $R''$ of expected size $\xi_\tau$. The expected complexity of $\mathcal{A}(R'')$ is $O(\xi_\tau + (\xi_\tau/m_\tau)^2 X_\tau) = O(\xi_\tau)$, since $\xi_\tau < s$. We apply the same decomposition construction as in the preceding case, obtaining a new collection of $O(\xi_\tau)$ trapezoids, each of which is crossed by at most $m_\tau/\xi_\tau$ curves of $C^*_\tau$. These trapezoids are the cells $\tau_0$ of the cutting-cover of $\tau^*$.

This concludes the description of the preliminary covering of $\tau^*$ that is constructed only if $X_\tau = o(m_\tau^2)$. If $X_\tau = \Theta(m_\tau^2)$, we have $s = O(1)$ and the first case applies; we cover $\tau$ by a single $\tau_0$, which we take to be the entire $xy$-plane.

We now apply an additional decomposition step to each cell $\tau_0$ of this preliminary cutting. This decomposition consists of a recursively constructed hierarchical sequence of cuttings of the subset $C^*_{\tau_0}$ of those curves of $C^*_\tau$ that cross $\tau_0$, clipped to within $\tau_0$. This decomposition is somewhat reminiscent of the hierarchical cutting construction of Chazelle [6]. We begin by choosing a sufficiently large constant $\rho$, to be used throughout the construction. Put $m_{\tau_0} = |C^*_{\tau_0}|$.

**First level in the hierarchy.** We draw a random sample $Q$ of $\rho$ arcs of $C^*_{\tau_0}$ and consider all the faces of the planar arrangement $\mathcal{A}(Q)$ that contain components of $\partial\tau^*$. By the definition of $C_\tau$, the arcs of $C^*_{\tau_0}$ are contained within $\tau^*$, and thus each component of $\partial\tau^*$ lives in a single (not necessarily distinct) face of $\mathcal{A}(Q)$. We refer to such faces as the *external* faces of $\mathcal{A}(Q)$. Note also that, as defined, those faces are not confined within $\tau_0$ or within $\tau^*$. That is, $\partial\tau^*$ is not part of $\mathcal{A}(Q)$ and does not delimit any face of it. However, each component $\gamma$ of $\partial\tau^*$ bounds a connected component of the complement of $\tau^*$ which is fully disjoint from all the arcs of $Q$ (or of $C^*_{\tau_0}$ for that matter). See Figure 2.

For each external face $f$ of $\mathcal{A}(Q)$, we compute the two-dimensional vertical decomposition of $f$ into vertical pseudotrapezoids (see, e.g., [18]), which we refer to as *trapezoids* or *subcells*. With high probability (greater than, say, $1 - 1/\rho$), each resulting subcell $\sigma$ is crossed by at most $\frac{am_{\tau_0}}{\rho} \log \rho$ curves of $C^*_{\tau_0}$ for an appropriate absolute constant $a$ [9, 14]. As above, we assume that $Q$ is a sample that satisfies this property. For each connected component $\gamma$ of $\partial\tau^*$, the face $f_\gamma$ of $\mathcal{A}(Q)$ that contains $\gamma$ consists of $O(\rho\beta_q(\rho))$ subcells [18], so the total number of crossings between the arcs of $C^*_{\tau_0}$ and these subcells is $O(m_{\tau_0}\beta_q(\rho)\log\rho)$. Let $\kappa_{\tau_0}$ denote the number of distinct external faces of $\mathcal{A}(Q)$. Then we get a total of $O(\kappa_{\tau_0}\rho\beta_q(\rho))$ external trapezoids,[1] and the total number of crossings between the arcs of $C^*_{\tau_0}$ and these subcells is $O(\kappa_{\tau_0}m_{\tau_0}\beta_q(\rho)\log\rho)$.

An obvious upper bound on $\kappa_{\tau_0}$ is $1+h_{\tau_0}$, where $h_{\tau_0}$ denotes the number of internal connected components of $\partial\tau^*$ that are fully contained in $\tau_0$ (boundary components that cross $\partial\tau_0$ all lie in the single unbounded face of $\mathcal{A}(Q)$), but we will use in the following analysis a more refined bound. The need for a refined analysis comes from the observation that, at this initial stage of the hierarchy, the total number of faces of $\mathcal{A}(Q)$ is only a constant (at most $O(\rho^2)$), whereas $h_{\tau_0}$ can be much larger. Note that, trivially,

$$\sum_{\tau_0} h_{\tau_0} \le h_\tau \le \xi_\tau. \tag{2}$$

We also have $h_\tau = O(r)$, because we can charge each internal component of $\partial\tau^*$ either to a complete connected component of an intersection curve between the surface of $R$ forming the floor of $\tau$ with another surface in $R$, to a similar intersection component involving the surface forming the ceiling of $\tau$, or to a complete connected component of the silhouette of some surface of $R$ (which is completely contained in the interior of $\tau$), and the overall number of such components is clearly $O(r)$. In fact, applying this analysis to all the cells $\tau$ of $\mathcal{A}_1$ together, we obtain the following bound, which is crucial for our analysis:

$$\sum_{\tau\in\mathcal{A}_1} h_\tau = O(r^2). \tag{3}$$

In addition to decomposing the external faces as described above, we also partition the remainder of $\mathcal{A}(Q)$ (its *internal portion*) into vertical trapezoids. In doing so, we erase all the edges of $\mathcal{A}(Q)$ that are contained in the interior of the internal portion, and retain only the edges that also bound the external faces. Thus the number of trapezoids into which the internal portion is partitioned is also $O(\kappa_{\tau_0}\rho\beta_q(\rho))$. The total number of crossings between the arcs of $C^*_{\tau_0}$ and these internal subcells is $O(\kappa_{\tau_0}m_{\tau_0}\rho\beta_q(\rho))$. (Here we can no longer claim that each internal trapezoid is crossed by only a small number of curves, because it is not necessarily disjoint from the sampled curves in $Q$, so this bound is larger than the bound claimed for external trapezoids by nearly a factor of $\rho$.)

**Second level in the hierarchy.** We now apply a second partitioning step[2] within each external trapezoid $\sigma$ that has a nonempty intersection with $\partial\tau^*$. (All

[1]The number of external trapezoids is proportional to the combined complexity of the external faces. In general, better bounds are known for the complexity of $\kappa_{\tau_0}$ faces in an arrangement of $\rho$ curves (see, e.g., [8]), but the cruder bound that we use suffices for our purposes.

[2]To help the reader follow the construction, we present the second stage explicitly and separately, even though it is a special case of the general recursive step, described later. As a matter of fact, it is also similar to the first-level partitioning.

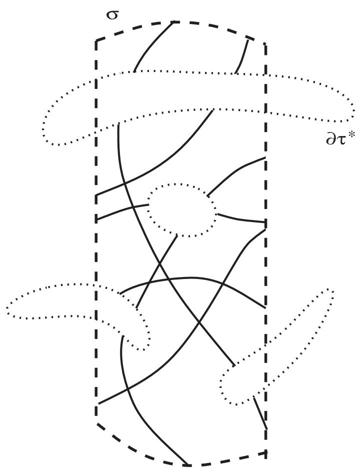


FIG. 3. *An external trapezoid $\sigma$ (dashed), the portions of $\partial\tau^*$ that meet $\sigma$ (dotted), and the arcs in $C^*_\sigma$ (solid).*

other external and internal trapezoids are not decomposed any further.) Let $C^*_\sigma$ denote the set of connected components of the intersections of the curves in $C^*_{\tau_0}$ with $\sigma$. As in the preceding step, $\sigma$ is not necessarily contained in $\tau^*$; however, each arc in $C^*_\sigma$ lies fully in $\sigma \cap \tau^*$. See Figure 3.

We draw a random sample $Q_\sigma$ of $\rho$ curves of $C^*_\sigma$ and compute all the faces of the planar arrangement $\mathcal{A}(Q_\sigma)$ that contain components of $\partial\tau^*$. As above, each component of $\partial\tau^*$ lives in a single ("external") face of $\mathcal{A}(Q_\sigma)$. Again, those faces are not necessarily distinct. This time, however, all external faces, with the exception of the unbounded one, are confined to within $\sigma$. Boundary components $\gamma$ of $\partial\tau^*$ that intersect $\sigma$ are of two types: those that are fully contained in the interior of $\sigma$, and those that cross $\partial\sigma$. All components $\gamma$ of the second type lie in the same (unbounded) face of $\mathcal{A}(Q_\sigma)$. Let $h_\sigma$, $\kappa_\sigma$ denote, respectively, the number of components $\gamma$ of the first type, and the number of distinct external faces of $\mathcal{A}(Q_\sigma)$. Clearly, $\kappa_\sigma \le 1 + h_\sigma$, and $\sum_\sigma h_\sigma \le h_\tau$ (where the sum extends over all $\sigma$ and all $\tau_0$). Again, however, we will have to use a more refined bound for $\kappa_\sigma$ in what follows.

For each external face $f$ of $\mathcal{A}(Q_\sigma)$, we compute the two-dimensional vertical decomposition of $f$. With high probability (larger than $1 - 1/\rho$), each resulting subcell $\sigma'$ is crossed by at most

$$\left(\frac{a \log \rho}{\rho}\right)^2 m_{\tau_0}$$

curves of $C^*_\sigma$, and, as above, we assume that $Q_\sigma$ is a sample that does satisfy this property. For each connected component $\gamma$ of $\partial\tau^*$ that meets $\sigma$, the face $f_\gamma$ of $\mathcal{A}(Q_\sigma)$ that contains $\gamma$ consists of $O(\rho\beta_q(\rho))$ subcells. Summing over all boundary components of $\partial\tau^*$ that meet $\sigma$, we get a total of $O(\kappa_\sigma \rho\beta_q(\rho))$ external trapezoids, and the total number of crossings between the arcs of $C^*_\sigma$ and these subcells is

$$O(\kappa_\sigma m_{\tau_0} \beta_q(\rho) \log^2 \rho/\rho).$$

Summing these bounds over all external trapezoids $\sigma$, we obtain bounds for the overall number of external trapezoids in the second hierarchical partitioning step and the total number of crossings between arcs in $C^*_{\tau_0}$ and these trapezoids. These bounds are, respectively,

$$\sum_{\sigma} O(\kappa_\sigma \rho \beta_q(\rho)) \tag{4}$$

and

$$\sum_{\sigma} O(\kappa_\sigma m_{\tau_0} \beta_q(\rho) \log^2 \rho/\rho),$$

where these sums are over all external trapezoids $\sigma$ in $\mathcal{A}(Q)$.

As above, we also partition the remaining internal portions of the arrangements $\mathcal{A}(Q_\sigma)$, over all trapezoids $\sigma$, into vertical trapezoids, using, as above, only the edges and vertices of these internal portions that bound also the external portions. Thus, the overall number of internal trapezoids is also bounded by (4), and the total number of crossings between arcs in $C^*_{\tau_0}$ and these internal trapezoids is at most

$$\sum_{\sigma \text{ an external trapezoid in } \mathcal{A}(Q)} O(\kappa_\sigma m_{\tau_0} \beta_q(\rho) \log \rho).$$

**Recursive construction of the hierarchy.** The above process is repeated recursively, each recursion stage refining the decomposition inside those "external" trapezoids constructed in the previous stage that are still crossed by (or contain) boundary components of $\partial\tau^*$. Let $j = j_{\tau_0}$ be the smallest integer such that

$$\rho^j \geq \xi_\tau / s.$$

We stop the recursive decomposition process after $j$ steps. In particular, if $\xi_\tau < s$, there is no recursion, and $\tau_0$ remains intact. Otherwise, we have $\rho^j = \Theta(\xi_\tau/s)$. Let us for now consider only the (much more involved) case $\xi_\tau \geq s$.

By an appropriate extension of the preceding arguments, the overall number of external and internal trapezoids produced in the $i$th step, for any $i = 1, \ldots, j$, is at most

$$\sum_{\sigma \text{ an external trapezoid in some } \mathcal{A}(Q_{\sigma'})} O(\kappa_\sigma \rho \beta_q(\rho)), \tag{5}$$

where $\sigma'$ is an external trapezoid constructed in the preceding $(i-1)$st step which intersects $\partial\tau^*$. With high probability (which we turn into certainty by choosing "good" samples $Q_{\sigma'}$), each external trapezoid constructed at the $i$th step is crossed by at most

$$O\left(\left(\frac{a \log \rho}{\rho}\right)^i m_{\tau_0}\right)$$

curves of $C^*_\tau$, and each such internal trapezoid is crossed by at most

$$O\left(\left(\frac{a \log \rho}{\rho}\right)^{i-1} m_{\tau_0}\right)$$

curves. Hence, the number of crossings between the arcs of $C^*_{\tau_0}$ and the external trapezoids is at most

$$\sum_\sigma O\left(\kappa_\sigma m_{\tau_0}\beta_q(\rho)\frac{a^i\log^i\rho}{\rho^{i-1}}\right),$$

and the number of crossings between the arcs of $C^*_{\tau_0}$ and the internal trapezoids is at most

$$\sum_\sigma O\left(\kappa_\sigma m_{\tau_0}\beta_q(\rho)\frac{a^{i-1}\log^{i-1}\rho}{\rho^{i-2}}\right), \tag{6}$$

where these sums are over all external trapezoids $\sigma$ in some $\mathcal{A}(Q_{\sigma'})$.

**Bounding the number of trapezoids.** We continue to assume in what follows that $\xi_\tau \geq s$; otherwise $\tau_0$ remains a single trapezoid. Let us analyze the number of trapezoids in more detail. Let $\gamma$ be a boundary component of $\partial\tau^*$. If at some step $i$, $\gamma$ crosses the boundary of some external trapezoid(s), it has no effect on the quantities $\kappa_\sigma$ from this step onward (inclusive). If on the other hand $\gamma$ remains confined to the interior of a single external trapezoid $\sigma$, then it may add 1 to $\kappa_\sigma$, but it will not affect $\kappa_{\sigma'}$, for any other external trapezoid $\sigma'$ produced at this step.

Elaborating this observation, we consider the tree $\mathcal{T}$ of all external trapezoids as they are generated during the recursive process. The root of the tree is $\tau_0$, and the children of each external trapezoid $\sigma$ are the external trapezoids that are constructed in the decomposition of $\sigma$. We say that a trapezoid is *pregnant* if it completely contains a component of $\partial\tau^*$ in its interior. Otherwise it is *empty*. An empty trapezoid can spawn at most $c\rho\beta_q(\rho)$ subtrapezoids (in a single decomposition step), for some constant $c$, whereas a pregnant trapezoid containing $t$ components of $\partial\tau^*$ in its interior can spawn as many as $(t+1)c\rho\beta_q(\rho)$ subtrapezoids, but no more than $c\rho^2$, which is the maximum number of trapezoids that can be generated in a single decomposition step (for simplicity, we use the same constant $c$ in both bounds). Moreover, the total number of pregnant trapezoids, over the entire tree, is only $O(h_{\tau_0})$.

The empty trapezoids are organized into subtrees, each rooted at some pregnant trapezoid. (If there are no pregnant trapezoids, the empty trapezoids comprise the entire tree $\mathcal{T}$, and the analysis becomes considerably simpler.) Consider such a subtree rooted at a pregnant trapezoid at depth $i$ (where the root of $\mathcal{T}$ is at depth 0). The degree of each node in the subtree is at most $c\rho\beta_q(\rho)$, so the size (or, more precisely, the number of leaves) of the subtree is at most $(c\rho\beta_q(\rho))^{j-i}$ (recall that $j$ is the depth of the entire recursion). We choose a threshold depth $k$, and distinguish between the cases $i \leq k$ and $i > k$. In the former case, the total number of trees whose roots are at depth $i$ is at most $c^i\rho^{2i}$, and their total size (i.e., number of leaves) is thus at most

$$c^i\rho^{2i}\cdot(c\rho\beta_q(\rho))^{j-i} = c^j\rho^{i+j}(\beta_q(\rho))^{j-i}.$$

Summing these bounds over all depths $i = 0,\ldots,k$, we obtain a total size of

$$O(c^j\rho^{k+j}(\beta_q(\rho))^{j-k})$$

trapezoids.

In the latter case ($i > k$), we bound the total number of trees whose roots are at depth greater than $k$ simply by $O(h_{\tau_0})$, and bound the size of any such subtree by $(c\rho\beta_q(\rho))^{j-k}$. Hence, the total size of these subtrees is at most

$$O(h_{\tau_0})\cdot(c\rho\beta_q(\rho))^{j-k}.$$

To fix the value of $k$, we first assume that $(c\rho^2)^j \le h_{\tau_0}$, set $k = j$, and note that only the case $i \le k$ remains relevant. The overall number of external trapezoids produced within $\tau_0$ under this assumption is

$$O(c^j \rho^{2j}) = O(h_{\tau_0}).$$

Assuming now that $(c\rho^2)^j > h_{\tau_0}$, we choose $k$ so that $(c\rho^2)^k = \Theta(h_{\tau_0})$, and assume that $\rho$ is a sufficiently large constant, as a function of a prescribed $\varepsilon > 0$, to conclude that the overall number of external trapezoids in this setting is

$$O(h_{\tau_0}^{1/2} \rho^{j(1+\varepsilon)}) = O\left( h_{\tau_0}^{1/2} \left( \frac{\xi_\tau}{s} \right)^{1+\varepsilon} \right).$$

By construction, the number of internal trapezoids has the same asymptotic upper bound.

Note that when $h_{\tau_0} = 0$, there is only one subtree, with $(c\rho\beta_q(\rho))^j = O((\xi_\tau/s)^{1+\varepsilon})$ trapezoids.

We sum the above two bounds over all cells $\tau_0$ (for the fixed first-stage cell $\tau$), use the Cauchy–Schwarz inequality and the facts that $\sum_{\tau_0} h_{\tau_0} \le h_\tau$ and that the number of trapezoids $\tau_0$ is $O(s)$, and cater to both cases $h_{\tau_0} > 0$ and $h_{\tau_0} = 0$ to conclude that the total number of trapezoids into which $\tau$ is partitioned is

$$\begin{aligned} (7) \qquad & O\left( \sum_{\tau_0} h_{\tau_0} \right) + O\left( \left( \frac{\xi_\tau}{s} \right)^{1+\varepsilon} \right) \cdot \sum_{\tau_0} \max\{1, h_{\tau_0}\}^{1/2} \\ &= O\left( \sum_{\tau_0} h_{\tau_0} \right) + O\left( \left( \frac{\xi_\tau}{s} \right)^{1+\varepsilon} (h_\tau + s)^{1/2} s^{1/2} \right) \\ &= O\left( h_\tau + (1 + h_\tau^{1/2}) \xi_\tau^{1+\varepsilon} \right). \end{aligned}$$

We now cater to the case $\xi_\tau < s$. In this case, $\tau$ is covered by $O(\xi_\tau)$ trapezoids $\tau_0$, and each of them remains intact, so the total number of trapezoids is $O(\xi_\tau)$, which is subsumed in the bound (7).

**Bounding the number of curve-cell crossings.** Next consider the bounds (6) on the number of curve-cell crossings, and we analyze them in more detail, using our tree representation of the external trapezoids. We continue to assume that $X_\tau \ge m_\tau$, and that $\xi_\tau \ge s$. For simplicity of exposition, we consider only crossings with external trapezoids, observing that at each step of the construction, the number of internal trapezoids has the same upper bound as the number of external trapezoids, and that, with high probability (which, as usual, we take to hold with certainty), the bound on the number of curves of $C_\sigma^*$ that cross an internal trapezoid is at most $\rho/(a \log \rho)$ times larger than the same bound for external trapezoids. Hence, up to this constant, the number of crossings with internal trapezoids has the same upper bound as the number of crossings with external trapezoids, so we concentrate only on bounding the latter quantity.

Consider our tree $\mathcal{T}$ of external trapezoids. As in the preceding analysis, we distinguish between the cases $h_{\tau_0} > 0$ and $h_{\tau_0} = 0$. We treat only the case $h_{\tau_0} > 0$; the other case is handled similarly, by replacing $h_{\tau_0}$ by 1. With high probability (which, as usual, we take to hold with certainty), an external trapezoid at depth $i$ is crossed by at most $(\frac{a \log \rho}{\rho})^i m_{\tau_0}$ curves of $C_{\tau_0}^*$. We fix a threshold value $k$ as above, taking also into consideration the case where $(c\rho^2)^j \le h_{\tau_0}$. Suppose first that $i \le k$. The number of external trapezoids at depth $i$ is at most $c^i\rho^{2i}$, so the overall number of curve-cell crossings with these trapezoids is at most $(ac\rho\log\rho)^i m_{\tau_0}$. Summing this over all depths $i = 0,\dots,k$, we get a total of $O((ac\rho\log\rho)^k m_{\tau_0})$ crossings. For both possible values of $k$, by the choices of $j$ and $\rho$, the above bound can be written as

$$O\left((ac^{1/2}\log\rho)^k(c^{1/2}\rho)^k m_{\tau_0}\right) = O\left(h_{\tau_0}^{1/2}(\xi_\tau/s)^\varepsilon m_{\tau_0}\right).$$

Consider next the case $i > k$ (which applies only when $(c\rho^2)^j > h_{\tau_0}$). The number of external trapezoids at depth $i$ can be estimated as follows. All these trapezoids belong to $O(h_{\tau_0})$ subtrees rooted at the pregnant trapezoids, or, if $h_{\tau_0} = 0$, to the entire tree $\mathcal{T}$. To maximize the number of our trapezoids, the subtrees should be rooted as close to the root of $\mathcal{T}$ as possible. By the choice of $k$, it is easily seen that this happens when all the pregnant nodes lie roughly at level $k$ of $\mathcal{T}$. Assuming this "worst-case" scenario, the number of external trapezoids at depth $i$ is at most

$$O\left((c\rho^2)^k \cdot (c\rho\beta_q(\rho))^{i-k}\right) = O\left(c^i\rho^{i+k}\beta_q^{i-k}(\rho)\right).$$

Since, with high probability (which we take to hold with certainty), each of these trapezoids is crossed by at most $(\frac{a\log\rho}{\rho})^i m_{\tau_0}$ curves of $C^*_{\tau_0}$, the total number of curve-cell crossings with these trapezoids is at most

$$(ac\beta_q(\rho)\log\rho)^i\,(\rho/\beta_q(\rho))^k\,m_{\tau_0}.$$

As in the preceding subcase, for both possible values of $k$, summing over all depths $i = k+1,\dots,j$, and using the choices of $j$ and $\rho$, this can be bounded by

$$O\left((ac\beta_q(\rho)\log\rho)^j\rho^k m_{\tau_0}\right) = O\left(h_{\tau_0}^{1/2}(\xi_\tau/s)^\varepsilon m_{\tau_0}\right).$$

Hence the total number of curve-cell crossings within $\tau_0$, taking also into account the case $h_{\tau_0} = 0$, is $O\left(\max\{1,h_{\tau_0}\}^{1/2}(\xi_\tau/s)^\varepsilon m_{\tau_0}\right)$.

We sum the bound just derived over all cells $\tau_0$ of $\mathcal{A}(R'')$, calibrate the value of $\varepsilon$ appropriately, and use the facts that the number of cells $\tau_0$ is $O(s)$, that $m_{\tau_0} \le m_\tau/s$, and that $s = \lceil m_\tau^2/X_\tau\rceil$. This yields the following overall bound:

$$\begin{aligned}
&O\left(\left(\frac{\xi_\tau}{s}\right)^\varepsilon\right)\cdot\sum_{\tau_0} O(m_{\tau_0}\max\{1,h_{\tau_0}\}^{1/2})\\
&= O\left(\frac{m_\tau\xi_\tau^\varepsilon}{s^{1+\varepsilon}}\cdot\left(\sum_{\tau_0}(1+h_{\tau_0})\right)^{1/2}\cdot s^{1/2}\right)\\
&= O\left(m_\tau\xi_\tau^\varepsilon\frac{(h_\tau+s)^{1/2}}{s^{1/2+\varepsilon}}\right)\\
&= O\left(\frac{\xi_\tau^\varepsilon}{s^\varepsilon}\left(m_\tau+\frac{m_\tau h_\tau^{1/2}}{s^{1/2}}\right)\right)\\
&= O\left((X_\tau^{1/2}h_\tau^{1/2}+m_\tau)\xi_\tau^\varepsilon\right) \qquad (8)
\end{aligned}$$

for any $\varepsilon > 0$.

If $\xi_\tau < s$, then $\tau_0$ remains intact and the number of crossings between curves and trapezoids within $\tau_0$ is thus $m_{\tau_0}$. We sum this over all $O(\xi_\tau)$ cells $\tau_0$ and use the fact that $m_{\tau_0} \le m_\tau/\xi_\tau$ for each $\tau_0$ to obtain the bound $O(m_\tau)$, which is subsumed in (8).

**The case $X_\tau < m_\tau$.** So far in the description of the second stage of the decomposition we have assumed that $X_\tau \geq m_\tau$. We now address the case $X_\tau < m_\tau$. By breaking each curve of $C^*_\tau$ at the points where it crosses other curves, we obtain a collection of pairwise openly disjoint curves, whose number is only $O(m_\tau)$. Assuming first that $\xi_\tau \leq m_\tau$, we now sample each (new) curve in $C^*_\tau$ with probability $\xi_\tau/m_\tau$, obtaining a random sample $R^*$ of expected size $\xi_\tau$. The expected complexity of the vertical decomposition of $\mathcal{A}(R^*)$ is thus also $O(\xi_\tau)$. By further refining the decomposition, we obtain a collection of $O(\xi_\tau)$ trapezoids, each crossed by at most $O(m_\tau/\xi_\tau)$ curves, for a total of $O(m_\tau)$ crossings between curves and cells. If $m_\tau < \xi_\tau$, we "sample" all curves in $C^*_\tau$ and construct the vertical decomposition of their arrangement. This yields $O(m_\tau) = O(\xi_\tau)$ trapezoids, each of which crosses no curve of $C^*_\tau$.

**Completion.** We now form the final two-dimensional decomposition by taking $\partial\tau^*$ into account. In the description below we address the more involved construction of the case $X_\tau \geq m_\tau$. The derived bounds can be shown to hold also when $X_\tau < m_\tau$ (with a significantly simpler analysis).

The final decomposition in the case $X_\tau \geq m_\tau$ is formed as follows. The hierarchy of trapezoids constructed so far is induced by various samples of (pieces of) curves from $C^*_\tau$. Let $\Gamma_\tau$ denote the collection of all curve portions that constitute the floors and ceilings of all these trapezoids. By construction, no two curve portions in $\Gamma_\tau$ intersect transversally. (Some pairs, constituting, e.g., floors of trapezoids that are nested in the hierarchy, may partially overlap; this has no effect on the analysis about to be presented.) Clearly, the number of trapezoids is $\Theta(|\Gamma_\tau|)$.

Consider now the union $\Gamma'_\tau$ of $\Gamma_\tau$ with the set of arcs forming $\partial\tau^*$. The arcs of $\Gamma'_\tau$ are also pairwise openly disjoint (recalling that the arcs of $\Gamma_\tau$ have been clipped at their points of intersection with $\partial\tau^*$). Form the vertical trapezoidal decomposition of $\Gamma'_\tau$. Using (7), the number of trapezoids in this decomposition is

$$O(|\Gamma'_\tau|) = O((1 + h_\tau^{1/2})\xi_\tau^{1+\varepsilon} + \xi_\tau + h_\tau) = O((1 + h_\tau^{1/2})\xi_\tau^{1+\varepsilon} + h_\tau).$$

We retain only those trapezoids that are fully contained in $\tau^*$ (the others are disjoint from $\tau^*$).

We next consider the number of crossings between the curves of $C^*_\tau$ and the new trapezoids. Each such crossing can be charged to a crossing of a curve $\gamma \in C^*_\tau$ with the boundary of a new trapezoid $\sigma$ (unless $\gamma$ is fully contained in $\sigma$; the number of such latter pairs is clearly at most $m_\tau$). If such a crossing occurs on the floor or ceiling of $\sigma$, then either it is also a crossing with the boundary of an old trapezoid, and is thus counted in (8), or it is an endpoint of a curve in $C^*_\tau$ (lying on a boundary component of $\partial\tau^*$), and the number of such endpoints is at most $2m_\tau$. If it occurs at a vertical wall erected from an endpoint (or a locally $x$-extreme point) $p$ of some arc in $\Gamma_\tau$, then the new wall is equal to or shorter than the old wall erected from $p$. Hence the number of such crossings is also upper bounded by (8). The only remaining case is a vertical wall erected from some vertex of $\partial\tau^*$ or from a locally $x$-extreme point on some arc of $\partial\tau^*$. The number of such walls is $O(\xi_\tau)$, and any such wall is fully contained in an old external trapezoid, and is thus crossed by at most

$$O((a \log\rho/\rho)^j m_{\tau_0}) = O((a \log\rho/\rho)^j (m_\tau/s))$$

curves of $C^*_\tau$. Hence the total number of crossings of this kind is (recall that $\rho^j = \Theta(\xi_\tau/s)$)

$$O(\xi_\tau (a \log\rho/\rho)^j (m_\tau/s)) = O(m_\tau \xi_\tau^\varepsilon)$$

for any $\varepsilon > 0$. This bound also takes care of the case $\xi_\tau < s$, and, as mentioned above, it also trivially holds when $X_\tau < m_\tau$.

The new decomposition is clearly a partition of $\tau^*$ into subcells (trapezoids) of constant description complexity. Each of these subcells is lifted vertically in the $z$-direction to within $\tau$, thereby obtaining a partition of $\tau$ itself. The collection of all these partitionings, over all cells $\tau$ of $\mathcal{A}_1$, constitutes our final decomposition.

Since each resulting (three-dimensional) cell has constant description complexity, it follows by the $\varepsilon$-net theory of Haussler and Welzl [14] that, with high probability, each of them is crossed by at most $\frac{a'n}{r}\log r$ surfaces of $S$, for an appropriate absolute constant $a' > 0$, so it is an $O((\log r)/r)$-cutting of $S$.

Lemma 2.2. (a) *The total number of cells of the above decomposition is* $O(r^{3+\varepsilon})$ *for any* $\varepsilon > 0$.

(b) *The total number of crossings between the curves of* $C$ *and these cells is* $O(mr^{1+\varepsilon})$ *for any* $\varepsilon > 0$.

*Proof.* (a) By (7), the number of cells is

$$O\left(\sum_{\tau\in\mathcal{A}_1}\left((1+h_\tau^{1/2})\xi_\tau^{1+\varepsilon}+h_\tau\right)\right) = O\left(r^2+\sum_{\tau\in\mathcal{A}_1}(1+h_\tau^{1/2})\xi_\tau^{1+\varepsilon}\right).$$

We analyze the quantity $O(\sum_{\tau\in\mathcal{A}_1}(1+h_\tau^{1/2})\xi_\tau^{1+\varepsilon})$. By Lemma 2.1(a), $\sum_\tau \xi_\tau^{1+\varepsilon} = O(r^{3+\varepsilon})$ for any $\varepsilon > 0$. This bound takes care of all cells for which $h_\tau = 0$. The number of cells with $h_\tau > 0$ is only $O(r^2)$. Moreover, the complexity of a single cell $\tau$ of $\mathcal{A}_1$ is only $O(r\beta_q(r))$. Indeed, such a cell has a fixed floor and a fixed ceiling, contained in two respective surfaces $\sigma^-, \sigma^+$ of $R$. We form a collection of curves consisting of the $xy$-projections of (i) the intersections of $\sigma^-$ and $\sigma^+$ with all the remaining surfaces of $R$, (ii) the silhouettes of the surfaces in $R$, and (iii) the curves in $R'$. We obtain a collection of $O(r)$ curves in the plane, and it is easily seen that $\tau^*$ is a cell of their arrangement. Hence the complexity of $\tau^*$, and thus of $\tau$, is $O(r\beta_q(r))$, as claimed (see [18] for details). Hence

$$\sum_{\tau\in\mathcal{A}_1} h_\tau^{1/2}\xi_\tau^{1+\varepsilon} = O\left(\left(\sum_{\tau\in\mathcal{A}_1} h_\tau\right)^{1/2}\cdot (r^2)^{1/2}\cdot r^{1+\varepsilon}\right) = O(r^{3+\varepsilon})$$

for any $\varepsilon > 0$, and this establishes (a).

(b) By (8) and the preceding discussion, the number of crossings is

$$\sum_\tau O\left((X_\tau^{1/2}h_\tau^{1/2}+m_\tau)\xi_\tau^{\varepsilon}\right)$$

for any $\varepsilon > 0$. Using (1) and (3), the Cauchy–Schwarz inequality, and Lemma 2.1(a,b), and recalibrating $\varepsilon$, this can be upper bounded by

$$O(r^\varepsilon)\cdot\left[\sum_\tau O(X_\tau^{1/2}h_\tau^{1/2})+\sum_\tau O(m_\tau)\right]$$

$$= O(r^\varepsilon)\cdot\left(\sum_\tau X_\tau\right)^{1/2}\cdot\left(\sum_\tau h_\tau\right)^{1/2} + O(mr^{1+\varepsilon}) = O(mr^{1+\varepsilon})$$

for any $\varepsilon > 0$. □

By replacing $r$ by $ar \log r$, for an appropriate absolute constant $a$, as discussed above, we obtain the following main result.

THEOREM 2.3. *Let $S$ be a set of $n$ surfaces in $\mathbb{R}^3$ of constant description complexity, and let $C$ be a set of $m$ curves in $\mathbb{R}^3$ of constant description complexity. Let $1 \le r \le \min\{m, n\}$ be a given parameter. Then there exists a $(1/r)$-cutting $\Xi$ of $S$ of size $O(r^{3+\varepsilon})$, for any $\varepsilon > 0$, such that the number of crossings between the curves of $C$ and the cells of $\Xi$ is $O(mr^{1+\varepsilon})$.*

*Remark* 1. We have ignored so far the algorithmic issue of constructing the cutting. However, the proof is constructive. Moreover, since at each step of the second decomposition stage we deal with samples of only $O(1)$ curves, the overall cost of the construction can be shown to be $O(nr^{2+\varepsilon} + mr^{1+\varepsilon})$ for any $\varepsilon > 0$. Recall that in the proof we assume that each random sample is a good sample. This can be algorithmically enforced by the standard approach of repeatedly sampling until a good sample is found. Since we use only constant-size samplings in the second decomposition stage, verifying that a sample is good is inexpensive. This approach increases the running time of the algorithm by a constant factor on expectation.

*Remark* 2. Theorem 2.3 bounds only the overall number of crossings between the curves and cells. A stronger result would be to show that, in addition, each cell of the cutting is crossed by $O(m/r)$ curves of $C$. We have not carried out this extension, but we believe that this stronger property can be achieved via a modified version of the preceding analysis.

**3. The complexity of a multiple zone.** Let $S$ and $C$ be as above. Define the zone $Z(C)$ of $C$ in $\mathcal{A}(S)$ to be the collection of all cells of $\mathcal{A}(S)$ that are crossed by at least one curve of $C$.

THEOREM 3.1. *The complexity of $Z(C)$ is $O(m^{1/2}n^{2+\varepsilon})$ for any $\varepsilon > 0$.*

*Proof.* Since the complexity of the entire arrangement is $O(n^3)$, the bound in the theorem is nontrivial only when $m = O(n^2)$, which is what we assume in the proof. Fix a parameter $r$, and construct a $C$-sensitive $(1/r)$-cutting of $\mathcal{A}(S)$, consisting of $O(r^{3+\varepsilon})$ cells, each crossed by at most $n/r$ surfaces of $S$, so that the total number of crossings between these cells and the curves of $C$ is at most $O(m^{1+\varepsilon}r)$.

Fix a cell $\tau$ of the cutting. Let $S_\tau$ (resp., $C_\tau$) denote the set of surfaces of $S$ (resp., curves of $C$) that cross $\tau$, clipped to within $\tau$. The complexity of $Z(C) \cap \tau$ can be upper bounded as follows: First, the zone of a single curve in an arrangement of $N$ surfaces of constant description complexity is $O(N^{2+\varepsilon})$ for any $\varepsilon > 0$ [13]. Hence, the overall complexity of the $|C_\tau|$ separate zones of each of the curves in $C_\tau$ in $\mathcal{A}(S_\tau)$ is at most $O(|C_\tau||S_\tau|^{2+\varepsilon})$. In addition, portions of the boundary of the external cell of $\mathcal{A}(S_\tau)$ may also belong to $Z(C)$, because they may bound cells of $\mathcal{A}(S)$ that are crossed by curves of $C$ that do not cross $\tau$. The complexity of this external cell is $O(|S_\tau|^{2+\varepsilon})$. Hence, putting $m_\tau = |C_\tau|$, the overall complexity of $Z(C)$ is (we use the same $\varepsilon$ both in the bounds in Theorem 2.3 and for the bound on the complexity of the zone of a curve)

$$O\left(\sum_\tau (m_\tau + 1)\left(\frac{n}{r}\right)^{2+\varepsilon}\right) = O\left(\frac{mn^{2+\varepsilon}}{r} + n^{2+\varepsilon}r\right),$$

where we use Theorem 2.3 to infer that $\sum_\tau m_\tau = O(mr^{1+\varepsilon})$. Choosing $r = m^{1/2}$ completes the proof of the theorem. □

*Remark* 3. A lower bound for $Z(C)$ is $\Omega(m^{2/3}n^{5/3})$. To establish it, take a planar arrangement of $n/2$ lines that has $m$ distinct faces of overall complexity $\Theta(m^{2/3}n^{2/3})$. Lift each of these lines to a vertical plane in three dimensions, and add to the resulting arrangement $n/2$ additional horizontal planes. The resulting collection of $n$ planes is our set $S$. For the set $C$ of curves, take $m$ vertical lines, each intersecting the $xy$-plane at a point inside one of the $m$ marked faces. The complexity of the multiple zone $Z(C)$ is easily seen to be $\Theta(m^{2/3}n^{5/3})$.

**Acknowledgments.** We would like to thank Gábor Tardos for suggesting the problem, and thank him and János Pach for valuable discussions. We also deeply thank two anonymous referees for very helpful comments on the paper, which have led to considerable improvements. In addition, we are greatly indebted to one of the referees, who has pointed out a serious technical problem in an earlier version of the paper.

## REFERENCES

[1] P. K. Agarwal, J. Matoušek, and O. Schwarzkopf, *Computing many faces in arrangements of lines and segments*, SIAM J. Comput., 27 (1998), pp. 491–505.

[2] P. K. Agarwal and M. Sharir, *Arrangements and their applications*, in Handbook of Computational Geometry, J.-R. Sack and J. Urrutia, eds., North-Holland, Amsterdam, 2000, pp. 49–119.

[3] B. Aronov, M. Pellegrini, and M. Sharir, *On the zone of a surface in a hyperplane arrangement*, Discrete Comput. Geom., 9 (1993), pp. 77–186.

[4] M. de Berg, L. J. Guibas, and D. Halperin, *Vertical decompositions for triangles in* 3*-space*, Discrete Comput. Geom., 15 (1996), pp. 35–61.

[5] M. de Berg and O. Schwarzkopf, *Cuttings and applications*, Internat. J. Comput. Geom. Appl., 5 (1995), pp. 343–355.

[6] B. Chazelle, *Cutting hyperplanes for divide-and-conquer*, Discrete Comput. Geom., 9 (1993), pp. 145–158.

[7] B. Chazelle and J. Friedman, *A deterministic view of random sampling and its use in geometry*, Combinatorica, 10 (1990), pp. 229–249.

[8] K. Clarkson, H. Edelsbrunner, L. J. Guibas, M. Sharir, and E. Welzl, *Combinatorial complexity bounds for arrangements of curves and spheres*, Discrete Comput. Geom., 5 (1990), pp. 99–160.

[9] K. Clarkson, *New applications of random sampling in computational geometry*, Discrete Comput. Geom., 2 (1987), pp. 195–222.

[10] K. Clarkson and P. Shor, *Applications of random sampling in computational geometry*, II, Discrete Comput. Geom., 4 (1989), pp. 387–421.

[11] D. P. Dobkin and D. G. Kirkpatrick, *Fast detection of polyhedral intersection*, in Proceedings of the 9th International Colloquium on Automata, Languages, and Programming, Lecture Notes in Comput. Sci. 140, Springer-Verlag, Berlin, 1982, pp. 154–165.

[12] E. Ezra and M. Sharir, *Counting and representing intersections among triangles in three dimensions*, in Proceedings of the 20th ACM Symposium on Computational Geometry, ACM, New York, 2004, pp. 210–219.

[13] D. Halperin and M. Sharir, *Almost tight upper bounds for the single cell and zone problems in three dimensions*, Discrete Comput. Geom., 14 (1995), pp. 385–410.

[14] D. Haussler and E. Welzl, *Epsilon-nets and simplex range queries*, Discrete Comput. Geom., 2 (1987), pp. 127–151.

[15] V. Koltun and M. Sharir, *Curve-sensitive cuttings*, in Proceedings of the 19th ACM Symposium on Computational Geometry, ACM, New York, 2003, pp. 136–143.

[16] J. Matoušek, *Construction of $\epsilon$-nets*, Discrete Comput. Geom., 5 (1990), pp. 427–448.

[17] J. Matoušek, *Range searching with efficient hierarchical cuttings*, Discrete Comput. Geom., 10 (1993), pp. 157–182.

[18] M. Sharir and P. K. Agarwal, *Davenport–Schinzel Sequences and Their Geometric Applications*, Cambridge University Press, New York, 1995.

SIAM J. COMPUT. © 2005 Society for Industrial and Applied Mathematics
Vol. 34, No. 4, pp. 879–893

# LOW-DIMENSIONAL LINEAR PROGRAMMING WITH VIOLATIONS*

TIMOTHY M. CHAN†

**Abstract.** Two decades ago, Megiddo and Dyer showed that linear programming (LP) in two and three dimensions (and subsequently any constant number of dimensions) can be solved in linear time. In this paper, we consider the LP problem with at most $k$ *violations*, i.e., finding a point inside all but at most $k$ halfspaces, given a set of $n$ halfspaces. We present a simple algorithm in two dimensions that runs in $O((n+k^2)\log n)$ expected time; this is faster than earlier algorithms by Everett, Robert, and van Kreveld (1993) and Matoušek (1994) for many values of $k$ and is probably near-optimal. An extension of our algorithm in three dimensions runs in near $O(n + k^{11/4}n^{1/4})$ expected time. Interestingly, the idea is based on concave-chain decompositions (or covers) of the ($\leq k$)-level, previously used in proving combinatorial $k$-level bounds.

Applications in the plane include improved algorithms for finding a line that misclassifies the fewest among a set of bichromatic points, and finding the smallest circle enclosing all but $k$ points. We also discuss related problems of finding local minima in levels.

**Key words.** computational geometry, algorithms, LP

**AMS subject classifications.** 68U05, 68Q25, 68W20, 90C05

**DOI.** 10.1137/S0097539703439404

## 1. Introduction.

**1.1. Motivation: Outliers in geometric optimization.** Consider the following formulation of a line-fitting problem, well known in computational geometry: given a set of data points $(x_1, y_1), \ldots, (x_n, y_n)$ in the plane, find a line that minimizes its largest vertical distance to the points. The problem is equivalent to a linear program (LP) in three variables (the slope $m$, the intercept $b$, and the tolerance $\delta$), where the goal is to minimize $\delta$ subject to the constraints $-\delta \leq mx_i + b - y_i \leq \delta$ for $i = 1, \ldots, n$. By known methods [22, 30, 49, 50, 57, 60], the problem can thus be solved in $O(n)$ time.

Though efficient methods exist, whether this formulation is the "right" one for real applications is debatable, as the presence of an occasional faulty data point (a so-called outlier) can drastically change the optimum. Statisticians have looked at more robust formulations of the line-fitting problem, but here we try to address the issue of outliers from a different direction by asking the following natural computational questions, keeping the largest vertical distance as the fitness measure: Given a small integer $k \leq n$, how can we find the line that best fits all but $k$ of the given points? Or, given a prescribed tolerance $\delta$, how can we find the smallest integer $k$ such that a line fitting all but $k$ points exists?

Low-dimensional LP-type techniques can solve other common optimization problems. For example, finding the smallest enclosing circle of a planar point set (the standard 1-center problem) is an instance of three-dimensional (3-d) convex programming,

*Received by the editors December 31, 2003; accepted for publication (in revised form) October 25, 2004; published electronically May 12, 2005. A preliminary version of this paper appeared in *Proceedings of the 43rd IEEE Symposium on Foundations of Computer Science*, 2002. This work was supported in part by an NSERC research grant.
http://www.siam.org/journals/sicomp/34-4/43940.html

†School of Computer Science, University of Waterloo, Waterloo, Ontario N2L 3G1, Canada (tmchan@uwaterloo.ca).

and finding the smallest-area enclosing annulus of a planar point set (a circle-fitting problem) is an instance of four-dimensional (4-d) LP. Due to applications in statistical analysis and computational metrology, it is again natural to consider generalizations of these optimization problems that allow a small number $k$ of violations.

**1.2. LP with violations: The problem and background.** We can define the problem of *linear programming with at most k violations* in $d$ dimensions as follows:

> Given a set $H$ of $n$ closed halfspaces in $\mathbb{R}^d$, a linear objective function $f$, and a number $0 \leq k < n/2$, we want to minimize $f$ over the region
>
> $$I_k(H) = \{q \in \mathbb{R}^d \mid q \text{ lies outside at most } k \text{ halfspaces of } H\},$$
>
> or report that $I_k(H) = \emptyset$.

Since our interest is in geometric applications, we confine our discussion to small constant values of $d$. The problem for arbitrary dimensions is NP-complete (for example, see [7, 8]).

Note that $I_0(H)$, the intersection of all halfspaces, is the feasible region of the original LP problem. Following Matoušek [47], we call the special case of the problem in which $I_0(H) \neq \emptyset$ the *feasible* case. In the feasible case (where, by an affine transformation, we may assume that the halfspaces are all lower halfspaces or all upper halfspaces), the boundary of $I_k(H)$ is commonly called the *k-level*, and the 0-level, 1-level, ..., $k$-level collectively form the *$(\leq k)$-level*.

The simplest case, the two-dimensional (2-d) feasible problem, can be solved in near-linear time by a parametric or binary search [47, 56]; the current best time bound was obtained by this author [14] using randomization and almost matched the one-dimensional (1-d) $O(n + k \log k)$ bound.

The next simplest case, the general 2-d problem, has already baffled researchers. Everett, Robert, and van Kreveld [37] first investigated the problem in 1993; they proposed a simple approach that effectively explores the entire solution space by constructing the $(\leq k)$-level of the upper halfplanes and the $(\leq k)$-level of the lower halfplanes and "intersecting" the two structures, so to speak. As the $(\leq k)$-level has $O(nk)$ complexity [6, 40] in the plane, they solved the 2-d problem in $O(n \log n + nk)$ time. The approach can probably (though nontrivially) be extended to any fixed dimension, but the running time would be higher, since the $(\leq k)$-level has worst-case complexity $\Theta(n^{\lfloor d/2 \rfloor} k^{\lceil d/2 \rceil})$ in $\mathbb{R}^d$ [24] (in three dimensions, a near-$O(nk^2)$ algorithm was indeed obtained by Efrat, Lindenbaum, and Sharir [34]).

Shortly after, in 1994, Matoušek [47] proposed another simple approach; it works quite differently and is best described in the feasible case. The algorithm enumerates all local minima of $I_0(H), \ldots, I_k(H)$ by computing the minimum of $I_0(H)$ by LP and repeatedly removing a solution's defining halfspace and reoptimizing, to generate $I_1(H)$ minima from the $I_0(H)$ minimum, $I_2(H)$ minima from the $I_1(H)$ minima, and so on. As the $(\leq k)$-level has $O(k^d)$ local minima [51], the cost of the algorithm is dominated by the cost of $O(k^d)$ dynamic LP operations. The general 2-d problem can be "lifted" to a feasible 3-d problem, and with the appropriate data structures [53], these $O(k^3)$ operations can be carried out in $O(n \log n + k^3 \log^2 n)$ time. The first term has been lowered to $O(n \log k)$ by the author [11, 12]; the second term can probably be lowered slightly as well by adopting recent data structures for dynamic convex hulls [16] (though the particular LP queries needed were not explicitly considered in [16]).

Table 1

*Time bounds for LP with at most $k$ violations. (In this paper, $\varepsilon > 0$ denotes an arbitrarily small constant, and $c$ denotes a specific constant.)*

| Problem | Best previous result(s) | (Refs.) | New result |
|---|---|---|---|
| 2-d feasible | $O(n + k^{1-\varepsilon} n^{\varepsilon} \log n)$ | [56, 14] | |
| 2-d general | $O(n \log n + nk)$ | [37] | |
| | $O(n \log k + k^3 \log^2 n)$ | [47, 11] | $O((n + k^2) \log n)$ |
| 3-d feasible | $O(n \log k + k^3 n^{\varepsilon})$ | [47, 11] | $O(n \log n + k^2 \log^2 n)$ |
| 3-d general | $O(nk^2(\log n + \log^2(n/k)))$ | [34] | |
| | $O(n \log k + k^4 n^{\varepsilon})$ | [47, 11] | $O(n \log n + k^{11/4} n^{1/4} \log^c n)$ |
| 4-d feasible | $O(n \log k + k^{8/3} n^{2/3+\varepsilon} + k^4 n^{1/3+\varepsilon})$ | [47, 11] | $O(n \log n + k^{11/4} n^{1/4} \log^c n)$ |

Modulo these small improvements, the rough bounds of $O(nk)$ by Everett, Robert, and van Kreveld (for large $k \geq \sqrt{n}$) and $O(n + k^3)$ by Matoušek (for small $k \leq \sqrt{n}$) have surprisingly remained the record for the general 2-d problem for about eight years. There seems no natural way to combine the two approaches to get a uniform time bound for all $k$. Concerning his algorithm, Matoušek [47] wondered whether it is possible to generate the local minima of $I_k(H)$ directly, without going through $I_0(H), I_1(H), \ldots$, since the number of local minima in the $k$-level is only $O(k^{d-1})$ [23, 52]. In particular, for the general 2-d problem, is it possible to bypass intermediate "infeasible" minima (defined by triples after the lifting) and generate only solutions (vertices) defined by pairs of halfplanes, as in Everett, Robert, and van Kreveld's algorithm? To put it bluntly, can Matoušek's 2-d algorithm be made more "planar"?

With the status of the 2-d problem unresolved, our understanding of LP with violations in higher dimensions is even more tentative. A suspected lower bound for the general $d$-dimensional problem (for $k \ll n/2$) is $\Omega(n + k^d)$ because of the following argument: Given a collection of $N$ hyperplanes, the problem of detecting affine degeneracy, i.e., a subset of $d+1$ hyperplanes intersecting at a common point, is conjectured to require $\Omega(N^d)$ time. (See [36] for a proof in a restricted model; for $d = 2$, the problem is so-called 3SUM-*hard* [38].) This problem can be reduced to LP with violations for any $n \geq 2N$ and $k = N - d - 1$; just take the $2N$ lower and upper (closed) halfspaces defined by the $N$ hyperplanes together with $n - 2N$ copies of the halfspace $x \leq M$ for a sufficiently large $M$.

**1.3. Main results.** Table 1 summarizes the previous results and our new results. We focus on the general case since, as we will demonstrate in section 4, the feasible problem in $d$ dimensions reduces to the general problem in $d-1$ dimensions by parametric or randomized search.

Our main result, presented in section 2, is an algorithm for the general 2-d problem that runs in $O((n+k^2) \log n)$ expected time (randomization is used in just one step of the algorithm). This algorithm not only simultaneously improves Everett, Robert, and van Kreveld's and Matoušek's algorithms for most values of $k$ (between $n^{\varepsilon}$ and $n/\log n$) but also essentially settles the complexity of the 2-d problem up to a logarithmic factor, assuming that the conjectured lower bound holds. The algorithm, like Matoušek's, actually generates all $O(k^2)$ local minima of $I_0(H), \ldots, I_k(H)$. The basic algorithm uses simpler data structures than Matoušek's and is arguably simpler than Everett, Robert, and van Kreveld's as well. The approach is noteworthy: although like Everett, Robert, and van Kreveld we will use the $(\leq k)$-level of the lower/upper halfplanes, we will not build the entire $(\leq k)$-level (which has size $O(nk)$) but instead will work with its "concave/convex-chain decomposition" (which involves only

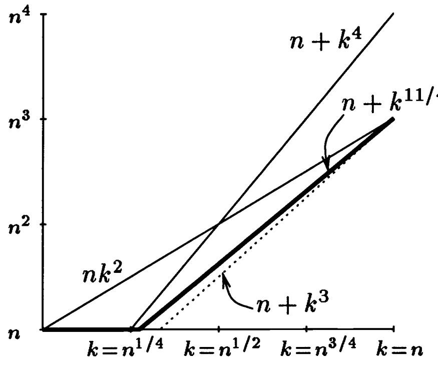


FIG. 1. *Comparison of time bounds for the general* 3*-d problem.*

a small $O(k)$ number of chains of total size $O(n)$). The concave-chain decomposition idea was instrumental in the recent breakthroughs on $k$-level complexity and related combinatorial problems [1, 27, 59]; because concave chains and convex polygons enjoy nice computational properties, we demonstrate that this very idea has algorithmic applications as well.

In section 3, we use similar ideas to obtain an $O(n \log n + k^3 \log^2 n)$ expected time bound for generating an $O(k^3)$-size superset of the local minima of $I_0(H), \dots, I_k(H)$ in three dimensions. The generalization is not trivial, as an analogous concave-surface decomposition of the ($\leq k$)-level in three dimensions is not known (but see [43] for a variant); nevertheless, we prove here that a small concave-surface *cover* always exists and can be computed efficiently. Unfortunately, unlike the 2-d algorithm, the 3-d algorithm does not automatically filter out the local minima of $I_0(H), \dots, I_k(H)$ from the superset. This filtering step turns out to be the computational bottleneck. Sophisticated static data structures for range searching are required to yield our final expected time bound of $O(n \log n + k^{11/4} n^{1/4} \log^c n)$. Although this does not quite approach the suspected lower bound $\Omega(n + k^3)$, it is not far off (see Figure 1) and in particular improves both Efrat, Lindenbaum, and Sharir's near-$O(nk^2)$ bound and Matoušek's near-$O(n + k^4)$ bound (the latter requires sophisticated dynamic data structures).

For very small $k = O(n^\varepsilon)$, Matoušek's algorithm can be made to run in $O(n \log k)$ time, as shown by the author [11, 12]. So, by combining two algorithms, we can automatically replace all $\log n$ factors with $\log k$ in the time bounds.

**1.4. Applications.** We can use our algorithms to find the smallest $k$ such that $I_k(H) \neq \emptyset$ within the same time bounds: Given an upper bound $K \geq k$, we can generate all local minima of $I_0(H), \dots, I_K(H)$ and thus identify the value of $k$. We can "guess" an upper bound by a standard trick (as in [12]); for example, in two dimensions, by trying $K = \sqrt{n}, 2\sqrt{n}, 4\sqrt{n}, \dots$, the total running time can be bounded by a geometric series and remains $O((n + k^2) \log n)$.

The 2-d algorithm can also be modified for convex programming with nearly the same running time, provided that a linear objective function is used, as explained in the remarks in section 2. The sample applications below thus follow immediately from these observations (all the bounds except the last are probably near-optimal for all $k \ll n/2$).

- Given $n$ red/blue points in the plane, we can find a line $\ell$ that minimizes $k$, the total number of red points above $\ell$ and blue points below $\ell$, in $O((n+k^2)\log n)$ expected time. (This was the original dual problem considered by Everett, Robert, and van Kreveld [37].)

- Given $n$ points in the plane and a fixed value $\delta$, we can find a line $\ell$ that minimizes $k$, the number of points at a vertical distance exceeding $\delta$ from $\ell$, in $O((n+k^2)\log n)$ expected time.
- Given $n$ points in the plane and a fixed convex set $C$ of constant complexity, we can translate $C$ to minimize $k$, the number of points outside $C$, in $O((n\beta(n)+k^2)\log n)$ expected time, where $\beta(\cdot)$ is a slow-growing, inverse-Ackermann-like function.
- Given $n$ points in the plane and a fixed value $\delta$, we can find an annulus $A$ of area $\delta$ that minimizes $k$, the number of points outside $A$, in $O(n\log n + k^{11/4}n^{1/4}\log^c n)$ expected time.

We can also use our algorithms to solve feasible LPs with violations in one dimension higher, as explained in section 4. The applications below follow:

- Given $n$ points in the plane and a number $k$, we can find the line that minimizes its largest vertical distance to all but $k$ points in $O(n\log n + k^2\log^2 n)$ expected time.
- Given $n$ points in the plane and a number $k$, we can find the smallest circle enclosing all but $k$ points in $O(n\beta(n)\log n + k^2 n^{\varepsilon})$ expected time. (This was the motivating problem considered by Matoušek [47]. Note that, in contrast, the related problem of finding the smallest circle enclosing $k$ points is believed to have complexity near $\Theta(nk)$ [26, 35, 42, 46].)
- Given $n$ points in the plane and a number $k$, we can find the minimum-area annulus enclosing all but $k$ points in $O(n\log n + k^{11/4}n^{1/4}\log^c n)$ expected time.

In the feasible case, several researchers have explored the problem of finding all local minima of the $k$-level and the $(\leq k)$-level.

- By specializing our 2-d algorithm in section 2 to the feasible case, we can immediately find all $O(k^2)$ local minima of the 2-d $(\leq k)$-level in $O((n+k^2)\log n)$ expected time, thus improving the previous $O(n\log n + k^2\log^{3/2} n)$ bound [16, 47].
- As shown in the appendix, we can also find all $O(k)$ local minima of the 2-d $k$-level in $O((n+(nk)^{3/5})\log^{O(1)} n)$ expected time, improving Katoh and Tokuyama's recent $O((n+(nk)^{2/3})\log^{O(1)} n)$ bound [44].

Again, all $\log n$ factors above can be replaced by $\log k$ by switching to Matoušek's algorithm when $k = O(n^{\varepsilon})$.

## 2. A 2-d algorithm to find all local minima of $I_0(H), \ldots, I_k(H)$.

Let $L^-$ and $L^+$ be the sets of lines bounding the given lower and upper halfplanes, respectively. Without loss of generality, assume that the objective is to minimize the $x$-coordinate. For simplicity, assume also that the input is in general position. The outline of our 2-d algorithm is remarkably simple:

1. Form $O(k)$ concave chains whose union covers (i.e., contains) the $(\leq k)$-level of the lower halfplanes; the polygonal chains are made from lines in $L^-$, are nonoverlapping (i.e., they intersect only at vertices), and have $O(n)$ total size (see Figure 2(a)). Similarly, form $O(k)$ convex chains for the lines in $L^+$.
2. For each pair of concave and convex chains, compute their left and right intersection points (if they exist). Let $S$ be the $O(k^2)$ left intersection points.

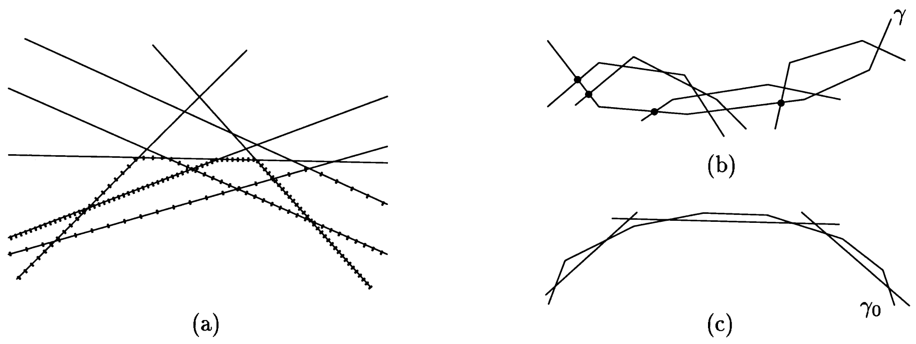


FIG. 2. (a) *The* $(\leq 2)$*-level of a set of lower halfplanes is covered by three concave chains (in dotted lines); in this example, the cover happens to be a decomposition.* (b) *Step* 3*: Counting concave chains below each point* $p \in S$ *(in black) on a convex chain* $\gamma$ *becomes an interval counting problem.* (c) *Step* 1*, second option: Forming concave chains becomes an interval coloring problem.*

3. For each $p \in S$, determine

$$k^-(p) = \min\{k+1, \text{number of lines of } L^- \text{ strictly below } p\}, \text{ and}$$
$$k^+(p) = \min\{k+1, \text{number of lines of } L^+ \text{ strictly above } p\}.$$

If $k^-(p) + k^+(p) \leq k$, then report $p$ as a local minimum of $I_{k^-(p)+k^+(p)}(H)$.

To check the correctness of the algorithm, we just have to observe that all local minima of $I_0(H), \ldots, I_k(H)$ are contained in $S$; each such local minimum lies on both the $(\leq k)$-level of the upper halfplanes and the $(\leq k)$-level of the lower halfplanes and thus is at the intersection of a concave and a convex chain.

We now explain how to implement the three steps, in order of difficulty, so that the total expected running time is $O((n + k^2)\log n)$.

**Step 2: Computing $S$.** The intersection of a concave and a convex chain can be found in logarithmic time by known binary-search algorithms (for example, see [20, 28]). So step 2 can be carried out in $O(k^2 \log n)$ time.

**Step 3: Computing $k^-(p)$ and $k^+(p)$.** By symmetry, it suffices to consider the computation of the $k^-(p)$ values. Observe that $k^-(p) = \min\{k + 1, \text{number of concave chains strictly below } p\}$: if $p$ is on the $(\leq k)$-level of the lower halfplanes, the number of lines of $L^-$ strictly below $p$ is the equal to the number of concave chains strictly below $p$, since the concave chains cover the $(\leq k)$-level and are nonoverlapping; if $p$ is strictly above the $(\leq k)$-level, both numbers exceed $k + 1$.

Fix a convex chain $\gamma$. Each concave chain defines an open interval on $\gamma$ delimited by the chain's left and right intersection points with $\gamma$. (The interval can be half-infinite or empty.) Given a point $p$ lying on $\gamma$, a concave chain is strictly below $p$ iff the corresponding interval does not contain $p$. (See Figure 2(b).) Thus, computing $k^-(p)$ for all points $p \in S$ lying on $\gamma$ can be reduced to a 1-d counting problem: given $O(k)$ points and $O(k)$ intervals, count how many intervals contain each point. This 1-d problem can be solved in $O(k \log k)$ time by sorting and a linear scan. Repeating the process for each convex chain $\gamma$ gives $k^-(p)$ for all points $p \in S$ in $O(k^2 \log k)$ time.

**Step 1: Constructing the concave/convex chains—first option.** By symmetry, it suffices to consider the computation of the concave chains. Pick a random integer $k' \in [k, 2k]$. Since the $k$-level, ..., $2k$-level have combined size $O(nk)$, the $k'$-level has expected size $O(n)$.

We use a standard idea to decompose the $(\leq k')$-level of the lower halfplanes into concave chains (for example, see [1]): Track the $k' + 1$ lowest lines of $L^-$ at a vertical sweep line as the sweep line moves from left to right. At $x = -\infty$, our $k' + 1$ chains start at the initial $k' + 1$ lowest lines. When the $(k' + 1)$st lowest line $\ell_1$ and the $(k' + 2)$nd lowest line $\ell_2$ are about to switch, the chain currently at $\ell_1$ will correspondingly turn right to follow $\ell_2$. Switches occur only at concave $k'$-level vertices, so the $k' + 1$ chains thus defined have total expected size $O(n)$ and can be formed in linear time once the $k'$-level has been constructed. (The example in Figure 2(a) is obtained this way.)

The $k'$-level can be computed by several output-sensitive algorithms, as surveyed in [15]. The simplest to implement is probably Basch, Guibas, and Hershberger's method using kinetic tournament trees [9]; for an expected $O(n)$-size output, the expected running time is $O(n\alpha(n) \log^2 n)$, where $\alpha(n)$ is the slow-growing inverse Ackermann function. The more complicated, randomized algorithms by Agarwal et al. [2, 15] and Har-Peled [41] are faster, with time bounds of $O(n\alpha(n)^2 \log n)$ and $O(n\alpha(n) \log n)$, respectively. Brodal and Jacob's recent announcement on dynamic convex hulls and kinetic heaps [10] implies the ultimate expected running time of $O(n \log n)$.

**Step 1: Constructing the concave chains—second option.** We now offer a different method for step 1 that also achieves $O(n \log n)$ expected time but has the advantage of being generalizable to three dimensions (as we will see in section 3).

This second option is based on Matoušek's shallow cutting lemma [45]. Ramos [54] (building on [2, 17]) has given an $O(n \log n)$ randomized algorithm to construct such a cutting in two and three dimensions. We restate the 2-d result in the following convenient form.

LEMMA 2.1. *Given $n$ lower halfplanes in $\mathbb{R}^2$, the $(\leq k)$-level can be covered by $O(n/k)$ cells, each intersecting $O(k)$ bounding lines. More specifically, the cells are taken from the vertical decomposition of the region underneath a concave chain $\gamma_0$ of size $O(n/k)$. The cells, the list of lines intersecting each cell, and $\gamma_0$ can all be constructed in $O(n \log n)$ expected time.*

*Proof.* Matoušek's shallow cutting lemma [45] and Ramos's algorithm [54] give a set $\Xi$ of cells (triangles) satisfying the first statement in $O(n \log n)$ expected time. To achieve the second statement, we modify the construction. Suppose that the maximum number of lines intersecting a cell is less than $bk$ for some constant $b$. Remove a cell from $\Xi$ if one of its vertices is above more than $(b + 1)k$ lines (because if this is true, the cell does not intersect the $(\leq k)$-level anyway); this condition can be tested in $O(\log n + k)$ time per cell, after preprocessing in $O(n \log n)$ time, by a known data structure for halfplane range reporting queries [21].

Now, let $\gamma_0$ be the boundary of the upper hull of the vertices of $\Xi$, and take the $O(n/k)$ cells (unbounded trapezoids) of its vertical decomposition, which is computable in $O((n/k) \log(n/k))$ time. Clearly, the new cells cover the $(\leq k)$-level. Since a line intersecting a cell must lie below one of its (two) vertices, the list of lines intersecting each new cell has at most $2(b + 1)k$ elements and can be generated in $O(\log n + k)$ time per cell, again by halfplane range reporting. ☐

Say that two lines are *compatible* in a region $R$ if no point in $R$ is above both lines. The problem of constructing the concave chains can be reduced to coloring the lines of $L^-$ so that lines of the same color are compatible in the $(\leq k)$-level of the lower halfplanes (or in any region covering the ($\leq k$)-level): Indeed, nonoverlapping concave chains can be formed by simply taking the lower envelope of lines of each color; the chains cover the ($\leq k$)-level, because given a point $p$ on the ($\leq k$)-level incident to some line $\ell$, $p$ cannot be above another line of the same color as $\ell$ and so must appear in the lower envelope of that color.

To solve this coloring problem, we invoke Lemma 2.1 to get a concave chain $\gamma_0$ above the ($\leq k$)-level. Each line of $L^-$ defines an interval (possibly half-infinite or empty) on $\gamma_0$ delimited by the line's left and right intersection points with $\gamma_0$; each interval can again be computed in logarithmic time by binary search. Two lines are compatible in the region underneath $\gamma_0$ iff their corresponding intervals are disjoint. (See Figure 2(c).) Thus, our problem is reduced to coloring of an interval graph.

Observe that the intervals have *depth* $O(k)$, i.e., each point $p$ on $\gamma_0$ is contained in at most $O(k)$ intervals: $p$ is contained in an interval iff $p$ lies above the corresponding line, but $p$ is above only $O(k)$ lines since each cell intersects $O(k)$ lines by construction. As is well known (see, e.g., [39]), the chromatic number of an interval graph is equal to the depth (also equal to the clique number), and an optimal coloring can be found in $O(n \log n)$ time by a standard greedy strategy (sequentially coloring the intervals in sorted order of left endpoints). This gives an $O(k)$-coloring, and thus a cover by $O(k)$ concave chains, in $O(n \log n)$ expected time.

We have thus proved the following theorem.

Theorem 2.2. *For $n$ halfplanes in $\mathbb{R}^2$, LP with at most $k$ violations can be solved in $O((n + k) \log n)$ expected time.*

*Remarks.* The construction [54] used in Lemma 3.1 is fairly involved. A much simpler alternative is to choose a random sample of $\frac{n}{2k}$ bounding planes and take the vertical decomposition of their lower envelope. This construction behaves the same "on average" but is only guaranteed to cover each vertex of the ($\leq k$)-level with constant probability (for example, as observed in [1]). The resulting algorithm would be Monte Carlo.

Degenerate cases can be handled by direct modifications of the algorithm or by general symbolic perturbation techniques [31, 32]. With the latter choice, we need to perturb the lines in $L^-$ upward and the lines in $L^+$ downward (as also suggested by Matoušek [47]), so that a vertex $v$ (lying on possibly more than two lines) violates at most $k$ constraints iff some point in the neighborhood of $v$ violates at most $k$ constraints after the perturbation.

The algorithm can be modified to work if the constraints $H$ are not halfplanes but convex sets of constant description complexity: For each convex set, we form a concave/convex $x$-monotone curve by taking its upper/lower boundary and attaching near-vertical downward/upward rays at the two endpoints. We let $L^-$ and $L^+$ instead be the sets of these concave and convex curves, respectively. Step 1 still works using the first option, since many of the $k$-level algorithms [2, 9, 15, 41] generalize to curves, with $\alpha(n)$ replaced by a similar slow-growing function $\beta(n)$ in the time bound. Step 2 generalizes, since concave/convex chains formed by concave/convex curves are still concave/convex. Step 3 requires a change, though. Given that $k^-(p), k^+(p) \leq k$, the number of constraints violated by $p$ is not necessarily $k^-(p) + k^+(p)$ but is rather $k^-(p) + k^+(p) - k_0(p)$, where $k_0(p)$ = number of convex sets whose $x$-projection does not contain $p$. Fortunately, computing the $k_0(p)$'s is also a 1-d interval counting problem and can be solved by sorting. The total expected running time is $O((n\beta(n) + k^2) \log n)$.

## 3. A 3-d algorithm to find all local minima of $I_0(H), \ldots, I_k(H)$.

Let $\Pi^-$ and $\Pi^+$ be the sets of planes bounding the given lower and upper halfspaces, respectively. Our 3-d algorithm proceeds similarly:

1. Form $O(k)$ concave surfaces whose union covers the $(\leq k)$-level of the lower halfspaces; the polyhedral surfaces are made from planes of $\Pi^-$, are nonoverlapping (i.e., they intersect only at zero-dimensional (0-d) or 1-d features), and have $O(n)$ total size. Similarly, form $O(k)$ convex surfaces for the planes of $\Pi^+$.
2. For each subset of at most three surfaces, find the minimum point in the intersection of the at most three convex polytopes bounded by these surfaces. Let $S$ be the $O(k^3)$ minima obtained.
3. For each $p \in S$, determine

$$k^-(p) = \min\{k+1, \text{number of planes of } \Pi^- \text{ strictly below } p\}, \text{ and}$$
$$k^+(p) = \min\{k+1, \text{number of planes of } \Pi^+ \text{ strictly above } p\}.$$

   If $k^-(p) + k^+(p) \leq k$, then report $p$ as a local minimum of $I_{k^-(p)+k^+(p)}(H)$.

Correctness follows since $S$ contains all local minima of $I_0(H), \ldots, I_k(H)$: each such local minimum $p$ lies on the $(\leq k)$-level of the lower halfspaces and of the upper halfspaces; in the neighborhood of $p$, its three defining halfspaces are part of up to three concave/convex surfaces; $p$ is the minimum point in the intersection of the corresponding polytopes.

We now explain how to implement each step.

**Step 2: Computing $S$.** If the polytopes are stored in hierarchical representations [29], which require $O(n)$ preprocessing time, then the minimum in the intersection of three (or fewer) convex polytopes can be found by an $O(\log^3 n)$ algorithm [33]. If the polytopes are instead stored in drum representations [55], which require $O(n \log n)$ preprocessing time [13], then the minimum can be found by an $O(\log^2 n)$ algorithm [13]. With the latter option, this step takes $O(k^3 \log^2 n)$ time.

**Step 1: Constructing the concave/convex surfaces—existence proof.** By symmetry, it suffices to consider the computation of the concave surfaces. Following the second option in section 2, we reduce the problem to coloring the planes of $\Pi^-$ so that two planes of the same color are compatible in the $(\leq k)$-level of the lower halfspaces (i.e., no point in the region is above both planes). We use the following 3-d version of Lemma 2.1.

LEMMA 3.1. *Given $n$ lower halfspaces in $\mathbb{R}^3$, the $(\leq k)$-level can be covered by $O(n/k)$ cells, each intersecting $O(k)$ bounding planes. More specifically, the cells are taken from the vertical decomposition of the region underneath a polyhedral concave surface $\gamma_0$ of size $O(n/k)$. The cells, the list of planes intersecting each cell, and $\gamma_0$ can all be constructed in $O(n \log n)$ expected time.*

*Proof.* As in the proof of Lemma 2.1, we apply Matoušek's shallow cutting lemma and Ramos's algorithm, which work in three dimensions. We modify the construction in the same way, this time using a randomized data structure for 3-d halfspace range reporting [17]. ☐

To solve the coloring problem, we invoke Lemma 3.1 to get a concave surface $\gamma_0$ lying above the $(\leq k)$-level. Each plane of $\Pi^-$ defines a set on $\gamma_0$ formed by intersecting its upper halfspace with $\gamma_0$. Two planes are compatible in the region underneath $\gamma_0$ iff the corresponding sets are disjoint.

As before, the sets have depth $O(k)$: a point $p$ on $\gamma$ lies above at most $O(k)$ planes since each cell intersects $O(k)$ planes by construction. We would like to infer that the intersection graph of planar convex sets of depth $O(k)$ is $O(k)$-colorable; however, this is not necessarily true in general. Fortunately, our sets are *pseudodisks*, i.e., the boundaries of each pair intersect at most twice: the intersection of two planes is a line, and a line intersects $\gamma_0$ at most twice. By a lemma of Sharir [58] (which extends a lemma of Pach), a 2-d arrangement of $n$ pseudodisks has $O(nk)$ intersections if their depth is $O(k)$. It follows that there exists a pseudodisk that intersects at most $O(k)$ other pseudodisks. Remove this pseudodisk and repeat. As a result, we obtain an acyclic orientation of the intersection graph so that the maximum in-degree $\delta$ is bounded by $O(k)$ (i.e., the *degeneracy* or *inductiveness* of the graph is $O(k)$). As is well known (see, e.g., [39]), the chromatic number is bounded by $\delta + 1$, and a corresponding coloring can be found by a standard greedy strategy. We conclude that an $O(k)$-coloring of the pseudodisks, and hence a cover by $O(k)$ concave surfaces, exists.

**Step 1: Constructing the concave surfaces—algorithmic proof.** Further ideas are needed to turn the above proof into an efficient algorithm, as we cannot afford to build the intersection graph of the pseudodisks and color sequentially. Here, we return to Matoušek's shallow cutting lemma (Lemma 3.1) and avoid Sharir's lemma (both were incidentally proven by random sampling arguments). Recall that we have $a|\Pi^-|/k$ cells, each intersected by at most $bk$ planes, for some constants $a$ and $b$. Call a plane *heavy* if it intersects more than $2ab$ cells and *light* otherwise (this is inspired by a similar definition of "bad" and "good" in [4]). At most half of the planes of $\Pi^-$ are heavy.

First, recursively color the heavy planes from the palette $\{1, \ldots, 4ab^2k\}$ so that planes of the same color are compatible in the $(\leq k)$-level of the corresponding heavy halfspaces (and thus in the $(\leq k)$-level of all lower halfspaces). Stop if the number of planes drops below $4ab^2k$. For each cell, maintain a dictionary of the colors used so far among the planes intersecting the cell.

Now, take each light plane $\pi$ (in any order). Randomly select a color from $\{1, \ldots, 4ab^2k\}$. Check that the color is not in the dictionary of any of the at most $2ab$ cells $\pi$ intersects. If so, give $\pi$ that color and update the dictionaries; otherwise, retry with another random selection. Since each of the at most $2ab$ dictionaries contains at most $bk$ colors, the probability of success in each trial is at least $1/2$, so this process takes expected $O(1)$ dictionary operations for each $\pi$. If the plane $\pi$ and a previous plane are incompatible in the region underneath $\gamma_0$, then some cell is intersected by both planes, and their colors are different by construction. So, after all light planes are processed, we have a correct $(4ab^2k)$-coloring.

The expected running time satisfies the recurrence $T(n) = T(n/2) + O(n \log n)$, which solves to $O(n \log n)$.

**Step 3: Computing $k^-(p)$ and $k^+(p)$.** By symmetry, it suffices to consider the computation of the $k^-(p)$ values. The geometry of concave/convex surfaces does not seem to help here, unlike in the 2-d algorithm, and we need to resort to more complicated techniques.

We wish to determine the number of planes strictly below each of $O(k^3)$ points. In dual space, this is the halfspace range counting problem. Since an upper limit $k+1$ is placed on the counts, one solution is to use known output-sensitive results, as given in [18, Theorem 6], to answer each halfspace range counting query in $O((1 + (nk^2/m)^{1/3})n^{\varepsilon})$ time after preprocessing in $O(mn^{\varepsilon})$ time. By setting the tradeoff parameter $m = \max\{k^{11/4}n^{1/4}, n\}$, the total cost of $O(k^3)$ queries becomes $O(n^{1+\varepsilon} + k^{11/4}n^{1/4+\varepsilon})$.

Alternatively, a more direct solution is to use Lemma 3.1. To compute $k^-(p)$, locate the cell $\Delta$ that contains $p$ (by 2-d point location), extract the list of the $O(k)$

planes intersecting $\Delta$, and count the number of planes in this list strictly below $p$. Recalling that $q$ halfspace range counting queries on a set of size $O(k)$ in three dimensions take $O(k \log k + (kq)^{3/4} \log^c k)$ time [3], we obtain the following bound on the total cost, where the $q_i$'s sum to $O(k^3)$:

$$O\left(n \log n + \sum_{i=1}^{O(n/k)} (kq_i)^{3/4} \log^c k\right) = O(n \log n + (n/k)^{1/4}(k \cdot k^3)^{3/4} \log^c k)$$
$$= O(n \log n + k^{11/4} n^{1/4} \log^c k).$$

Since range searching structures are used as a subroutine, the result is admittedly theoretical, but because the queries are off-line, (hierarchical) cuttings instead of partition trees are sufficient [3, 19] and the constant $c$ is small. (Without sophisticated data structures, one can still get a near-$O(n + k^4)$ time bound.)

We have thus proved the following theorem.

THEOREM 3.2. *For $n$ halfspaces in $\mathbb{R}^3$, LP with at most $k$ violations can be solved in $O(n \log n + k^{11/4} n^{1/4} \log^{O(1)} n)$ expected time.*

**4. On the feasible case.** We now solve the feasible LP problem with violations in $\mathbb{R}^d$ by reducing it to general LP with violations in $\mathbb{R}^{d-1}$.

Assume the objective is to minimize the first coordinate $x$. Let $\xi^*$ be the minimum $x$-coordinate in $I_k(H)$. First, find some $v_0 \in I_0(H)$ by LP. Since $I_k(H)$ is connected in the feasible case, given $\xi$ smaller than the $x$-coordinate of $v_0$, we can decide whether $\xi^* \leq \xi$ by testing whether $I_k(H)$ intersects the vertical hyperplane $x = \xi$ or, equivalently, whether $I_k(H_\xi) \neq \emptyset$, where $H_\xi$ is the set of $(d-1)$-d halfspaces formed by intersecting the halfspaces of $H$ with $x = \xi$.

**4.1. 3-d feasible LP with violations.** In the $d = 3$ case, the decision problem can therefore be solved by the 2-d algorithm of section 2 in $O((n + k^2) \log n)$ time. To solve the optimization problem, we can apply parametric search [48]. (We assume that the reader is familiar with this technique; if not, see [5] for more information.) Unfortunately, step 1 of our algorithm, the construction of the concave/convex chains, appears difficult to parallelize efficiently.

To circumvent this difficulty, we preprocess before parametric searching by constructing the 3-d concave/convex surface cover using step 1 of the algorithm of section 3, in $O(n \log n)$ expected time. If the bounding polytopes are stored in drum representations [55] via persistent search trees, as suggested in [13], we can retrieve a binary-searchable copy of the intersection of each surface with any vertical plane $x = \xi$ in logarithmic time. These 2-d slices form concave/convex chains covering the $(\leq k)$-level of the lower/upper halfplanes of $H_\xi$. Thus, for any given $\xi$, we can decide whether $I_k(H_\xi) = \emptyset$ in $O(k^2 \log n)$ time using steps 2 and 3 of the 2-d algorithm. These two steps are parallelizable in $O(\log n)$ time with $O(k^2)$ processors, by using the AKS sorting network for the interval counting problem in step 3 (the linear scan does not need to be parallelized because no more comparisons with $\xi$ are involved after sorting). Thus, $\xi^*$ can be found by Megiddo's parametric search [48], with Cole's improvement [25], in $O(k^2 \log^2 n)$ time. Including the preprocessing, the entire algorithm takes $O(n \log n + k^2 \log^2 n)$ expected time.

**4.2. 4-d feasible LP with violations.** Parametric search is more problematic in the $d = 4$ case. Not only is the decision algorithm (the 3-d algorithm of section 3) difficult to parallelize (because of step 1), but an efficient global preprocessing in four dimensions is not available to remedy the situation. Fortunately, the author's randomized optimization technique [14] is applicable (which uses the decision algorithm only as a black box) and yields the best result in this case.

LEMMA 4.1. *If LP with at most $k$ violations in $\mathbb{R}^{d-1}$ can be solved in $O(T(n,k))$ expected time, then feasible LP with at most $k$ violations in $\mathbb{R}^d$ can be solved in $O(T(n,k))$ expected time, provided that $T(n,k)/n^\varepsilon$ is monotone increasing in $n$.*

The $d = 2$ case of the above lemma is shown in [14, Theorem 5.2]. Since the proof in higher dimensions is essentially identical (using known constant-size cuttings in higher dimensions), we will not repeat the proof here.

By Lemma 4.1 and the algorithm of section 3, we can thus solve the 4-d feasible case in $O(n \log n + k^{11/4} n^{1/4} \log^c n)$ expected time.

THEOREM 4.2. *For $n$ halfspaces in $\mathbb{R}^3$ with a nonempty common intersection, LP with at most $k$ violations can be solved in $O(n \log n + k^2 \log^2 n)$ expected time.*

*For $n$ halfspaces in $\mathbb{R}^4$ with a nonempty common intersection, LP with at most $k$ violations can be solved in $O(n \log n + k^{11/4} n^{1/4} \log^{O(1)} n)$ expected time.*

*Remark.* A similar approach can be used to find the smallest circle enclosing all but $k$ points, given $n$ points $\{(a_i, b_i)\}_i$ in the plane. This problem is a variant of 3-d feasible LP with violations: the objective is now to minimize a convex function $x^2 + y^2 + z$, but the constraints $z \geq -2a_i x - 2b_i y + a_i^2 + b_i^2$ are still linear. The same technique for Lemma 4.1 reduces the problem to 2-d convex programming with violations (deciding whether there exists a point lying in all but $k$ planar convex sets), which can be solved in $O((n\beta(n) + k) \log n)$ expected time by the remark in section 2. The upper bound $T(n) = O(n\beta(n) \log n + kn^\varepsilon)$ can then be used.

(Alternatively, it might be possible to solve this problem by parametric search, but certain details are unclear, so we will not comment further.)

**5. Conclusions.** We have presented a new, simple approach to solving LPs with violations in two and three dimensions. The running time of our 2-d algorithm ($O((n + k^2) \log n)$) is almost optimal for all values of $k \ll n/2$ under a well-known conjecture; an open question is whether the time bound can be further improved to $O(n \log k + k^2)$. Our algorithm uses $O(n + k^2)$ space; another question is whether the storage requirement can be reduced to $O(n)$.

For the general 3-d problem, there is still a small gap between our time bound and the conjectured $\Omega(n + k^3)$ lower bound. For the general 4-d problem, our approach does not seem to work as well due to various reasons (for instance, we can no longer afford to construct concave/convex surfaces explicitly, since convex polytopes defined by $n$ hyperplanes in $\mathbb{R}^d$ may have $\Omega(n^{\lfloor d/2 \rfloor})$ size). Determining the complexity of LP with violations in dimension four and beyond thus remains a challenging problem.

**Appendix: Finding all local minima of $I_k(H)$ in the 2-d feasible case.** In the 2-d feasible case, Katoh and Tokuyama [44] recently showed that all $O(k)$ local $y$-maxima of the $k$-level of $n$ lower halfplanes can be enumerated in $O(n \log n + (nk)^{2/3} \log^{O(1)} n)$ time. In this appendix, we note that the time bound can be reduced to $O((n + (nk)^{3/5}) \log^{O(1)} n)$ using randomization. This result is independent of the rest of the paper and is of mainly theoretical interest, because in practice the $k$-level in two dimensions tends to have near-linear size, in which case the problem can be solved directly in near-linear time.

First, it is no loss of generality in the feasible case to assume that the objective is to maximize the $y$-coordinate and the halfplanes are all lower halfplanes: the first condition can be met by rotation; for the second condition, we can find a point $v_0 \in$

$I_0(H)$ by LP, make $v_0$ the origin by translation, and then transform each halfplane $ax + by \leq 1$ to the lower halfplane $y' \leq ax' - b$ by an affine map ($x' = -x/y$, $y' = -1/y$).

Our result is obtained by the following observation (which was also used in an algorithm in [15]): although the current best bound on size of the $k$-level is $O(nk^{1/3})$ by Dey [27], a random level nearby has lower complexity. Let $j$ be a parameter to be set later. Pick a random integer $j' \in [0, j]$ and construct the $(k - j')$-level $\mathcal{L}^-$ and the $(k + j')$-level $\mathcal{L}^+$. Since the $(k - j)$-level, …, $(k + j)$-level have combined size $O(nk^{1/3}j^{2/3})$ [27], $\mathcal{L}^-$ and $\mathcal{L}^+$ have expected size $O(n(k/j)^{1/3})$ and can therefore be constructed in $O(n(k/j)^{1/3} \log^{O(1)} n)$ expected time by known output-sensitive algorithms [15].

Among the $n$ bounding lines, track the subset formed by the $(k - j' + 1)$st, …, $(k + j' + 1)$st lowest lines at a vertical sweep line as the sweep line moves from left to right. Changes to the subset occur only at vertices of $\mathcal{L}^-$ and $\mathcal{L}^+$, and thus this process takes $O(n(k/j)^{1/3})$ time.

Now, divide the plane into $m = O(nk^{1/3}/j^{4/3})$ vertical slabs, each containing at most $j$ vertices of $\mathcal{L}^-$ and $\mathcal{L}^+$. Take each slab $\sigma$. A line intersecting the $k$-level within $\sigma$ must intersect $\mathcal{L}^-$ or $\mathcal{L}^+$, or be among the $(k - j' + 1)$st, …, $(k + j' + 1)$st lowest lines at the left/right wall of $\sigma$. Therefore, we can form a list $L_\sigma$ of $O(j)$ size that contains all lines involved in the $k$-level within $\sigma$. Within $\sigma$, the $k$-level of all lower halfplanes coincides with a level of the $O(j)$ lower halfplanes defined by $L_\sigma$. We can apply Katoh and Tokuyama's algorithm to find all $k_\sigma$ local $y$-maxima within $\sigma$ in $O(j \log j + (jk_\sigma)^{2/3} \log^{O(1)} j)$ time, because their algorithm is output-sensitive [44]. The total time over all slabs is given below, up to polylogarithmic factors, where the $k_\sigma$'s sum to $O(k)$:

$$\begin{aligned} O\left(mj + \sum_\sigma (jk_\sigma)^{2/3}\right) &= O(mj + m^{1/3}j^{2/3}k^{2/3}) \\ &= O(n(k/j)^{1/3} + n^{1/3}k^{7/9}j^{2/9}). \end{aligned}$$

Setting $j = \min\{n^{6/5}/k^{4/5}, k\}$ yields an expected time bound of $O((n + (nk)^{3/5}) \log^{O(1)} n)$.

*Remark.* If Dey's bound on the combined size of the $(k-j)$-level, …, $(k+j)$-level can be improved to $O(nk^\alpha j^{1-\alpha})$ for some constant $\alpha > 0$, then the time bound would work out to be $O((n + (nk^{3\alpha})^{1/(1+2\alpha)}) \log^{O(1)} n)$.

## REFERENCES

[1] P. K. Agarwal, B. Aronov, T. M. Chan, and M. Sharir, *On levels in arrangements of lines, segments, planes, and triangles*, Discrete Comput. Geom., 19 (1998), pp. 315–331.

[2] P. K. Agarwal, M. de Berg, J. Matoušek, and O. Schwarzkopf, *Constructing levels in arrangements and higher order Voronoi diagrams*, SIAM J. Comput., 27 (1998), pp. 654–667.

[3] P. K. Agarwal and J. Erickson, *Geometric range searching and its relatives*, in Discrete and Computational Geometry: Ten Years Later, B. Chazelle, J. E. Goodman, and R. Pollack, eds., AMS, Providence, RI, 1999, pp. 1–56.

[4] P. K. Agarwal and J. Matoušek, *Dynamic half-space range reporting and its applications*, Algorithmica, 13 (1995), pp. 325–345.

[5] P. K. Agarwal and M. Sharir, *Efficient algorithms for geometric optimization*, ACM Comput. Surv., 30 (1998), pp. 412–458.

[6] N. Alon and E. Győri, *The number of small semispaces of a finite set of points in the plane*, J. Combin. Theory Ser. A, 41 (1986), pp. 154–157.

[7] E. Amaldi and V. Kann, *The complexity and approximability of finding maximum feasible subsystems of linear relations*, Theoret. Comput. Sci., 147 (1995), pp. 181–210.

[8] E. Amaldi and V. Kann, *On the approximability of minimizing nonzero variables or unsatisfied relations in linear systems*, Theoret. Comput. Sci., 209 (1998), pp. 237–260.

[9] J. Basch, L. J. Guibas, and J. Hershberger, *Data structures for mobile data*, J. Algorithms, 31 (1999), pp. 1–28.

[10] G. S. Brodal and R. Jacob, *Dynamic planar convex hull*, in Proceedings of the 43rd IEEE Symposium on Foundations of Computer Science, IEEE Computer Society, Los Alamitos, CA, 2002, pp. 617–626.

[11] T. M. Chan, *Fixed-dimensional linear programming queries made easy*, in Proceedings of the 12th ACM Symposium on Computational Geometry, ACM, New York, 1996, pp. 284–290.

[12] T. M. Chan, *Output-sensitive results on convex hulls, extreme points, and related problems*, Discrete Comput. Geom., 16 (1996), pp. 369–387.

[13] T. M. Chan, *Deterministic algorithms for 2-d convex programming and 3-d online linear programming*, J. Algorithms, 27 (1998), pp. 147–166.

[14] T. M. Chan, *Geometric applications of a randomized optimization technique*, Discrete Comput. Geom., 22 (1999), pp. 547–567.

[15] T. M. Chan, *Remarks on $k$-Level Algorithms in the Plane*, 1999, manuscript.

[16] T. M. Chan, *Dynamic planar convex hull operations in near-logarithmic amortized time*, J. ACM, 48 (2001), pp. 1–12.

[17] T. M. Chan, *Random sampling, halfspace range reporting, and construction of ($\leq k$)-levels in three dimensions*, SIAM J. Comput., 30 (2000), pp. 561–575.

[18] T. M. Chan, *On enumerating and selecting distances*, Internat. J. Comput. Geom. Appl., 11 (2001), pp. 291–304.

[19] B. Chazelle, *Cutting hyperplanes for divide-and-conquer*, Discrete Comput. Geom., 9 (1993), pp. 145–158.

[20] B. Chazelle and D. P. Dobkin, *Intersection of convex objects in two and three dimensions*, J. ACM, 34 (1987), pp. 1–27.

[21] B. Chazelle, L. Guibas, and D. T. Lee, *The power of geometric duality*, BIT, 25 (1985), pp. 76–90.

[22] K. L. Clarkson, *Las Vegas algorithms for linear and integer programming when the dimension is small*, J. ACM, 42 (1995), pp. 488–499.

[23] K. L. Clarkson, *A bound on local minima of arrangements that implies the upper bound theorem*, Discrete Comput. Geom., 10 (1993), pp. 427–233.

[24] K. L. Clarkson and P. W. Shor, *Applications of random sampling in computational geometry*, II, Discrete Comput. Geom., 4 (1989), pp. 387–421.

[25] R. Cole, *Slowing down sorting networks to obtain faster sorting algorithms*, J. ACM, 34, (1987), pp. 200–208.

[26] A. Datta, H.-P. Lenhof, C. Schwarz, and M. Smid, *Static and dynamic algorithms for $k$-point clustering problems*, J. Algorithms, 19 (1995), pp. 474–503.

[27] T. K. Dey, *Improved bounds on planar $k$-sets and $k$-levels*, Discrete Comput. Geom., 19 (1998), pp. 373–382.

[28] D. P. Dobkin and D. G. Kirkpatrick, *Fast detection of polyhedral intersection*, Theoret. Comput. Sci., 27 (1983), pp. 241–253.

[29] D. P. Dobkin and D. G. Kirkpatrick, *Determining the separation of preprocessed polyhedra: A unified approach*, in Proceedings of the 17th International Colloquium of Automata, Languages, and Programming, Lecture Notes in Comput. Sci. 443, Springer-Verlag, New York, 1990, pp. 400–413.

[30] M. E. Dyer, *Linear time algorithms for two- and three-variable linear programs*, SIAM J. Comput., 13 (1984), pp. 31–45.

[31] H. Edelsbrunner and E. P. Mücke, *Simulation of simplicity: A technique to cope with degenerate cases in geometric algorithms*, ACM Trans. Graphics, 9 (1990), pp. 66–104.

[32] I. Emiris and J. Canny, *A general approach to removing degeneracies*, SIAM J. Comput., 24 (1995), pp. 650–664.

[33] D. Eppstein, *Dynamic three-dimensional linear programming*, ORSA J. Comput., 4 (1992), pp. 360–368.

[34] A. Efrat, M. Lindenbaum, and M. Sharir, *Finding maximally consistent sets of halfspaces*, in Proceedings of the 5th Canadian Conference on Computational Geometry, University of Waterloo, Waterloo, Ontario, Canada, 1993, pp. 432–436.

[35] A. Efrat, M. Sharir, and A. Ziv, *Computing the smallest $k$-enclosing circle and related problems*, Comput. Geom., 4 (1994), pp. 119–136.

[36] J. Erickson, *New lower bounds for convex hull problems in odd dimensions*, SIAM J. Comput., 28 (1999), pp. 1198–1214.

[37] H. Everett, J.-M. Robert, and M. van Kreveld, *An optimal algorithm for the ($\leq k$)-levels, with applications to separation and transversal problems*, Internat. J. Comput. Geom. Appl., 6 (1996), pp. 247–261.

[38] A. Gajentaan and M. H. Overmars, *On a class of $O(n^2)$ problems in computational geometry*, Comput. Geom., 5 (1995), pp. 165–185.

[39] M. C. Golumbic, *Algorithmic Graph Theory and Perfect Graphs*, Academic Press, New York, 1980.

[40] J. E. Goodman and R. Pollack, *On the number of $k$-subsets of a set of $n$ points in the plane*, J. Combin. Theory Ser. A, 36 (1984), pp. 101–104.

[41] S. Har-Peled, *Taking a walk in a planar arrangement*, SIAM J. Sci. Comput., 30 (2000), pp. 1341–1367.

[42] S. Har-Peled and S. Mazumdar, *Fast algorithms for computing the smallest $k$-enclosing disc*, in Proceedings of the 11th European Symposium on Algorithms, Lecture Notes in Comput. Sci. 2832, Springer-Verlag, New York, 2003, pp. 278–288.

[43] N. Katoh and T. Tokuyama, *Lovász's lemma for the three-dimensional $k$-level of concave surfaces and its applications*, Discrete Comput. Geom., 27 (2002), pp. 567–584.

[44] N. Katoh and T. Tokuyama, *Notes on computing peaks in $k$-levels and parametric spanning trees*, in Proceedings of the 17th ACM Symposium on Computational Geometry, ACM, New York, 2001, pp. 241–248.

[45] J. Matoušek, *Reporting points in halfspaces*, Comput. Geom., 2 (1992), pp. 169–186.

[46] J. Matoušek, *On enclosing $k$ points by a circle*, Inform. Process. Lett., 53 (1995), pp. 217–221.

[47] J. Matoušek, *On geometric optimization with few violated constraints*, Discrete Comput. Geom., 14 (1995), pp. 365–384.

[48] N. Megiddo, *Applying parallel computation algorithms in the design of serial algorithms*, J. ACM, 30 (1983), pp. 852–865.

[49] N. Megiddo, *Linear-time algorithms for linear programming in $R^3$ and related problems*, SIAM J. Comput., 12 (1983), pp. 759–776.

[50] N. Megiddo, *Linear programming in linear time when the dimension is fixed*, J. ACM, 31 (1984), pp. 114–127.

[51] K. Mulmuley, *Output sensitive construction of levels and Voronoi diagrams in $R^d$ of order 1 to $k$*, in Proceedings of the 22nd ACM Symposium on Theory of Computing, ACM, New York, 1990, pp. 322–330.

[52] K. Mulmuley, *Dehn-Sommerville relations, upper bound theorem, and levels in arrangements*, in Proceedings of the 9th ACM Symposium on Computational Geometry, ACM, New York, 1993, pp. 240–246.

[53] M. H. Overmars and J. van Leeuwen, *Maintenance of configurations in the plane*, J. Comput. System Sci., 23 (1981), pp. 166–204.

[54] E. Ramos, *On range reporting, ray shooting, and $k$-level construction*, in Proceedings of the 15th ACM Symposium on Computational Geometry, ACM, New York, 1999, pp. 390–399.

[55] M. Reichling, *On the detection of a common intersection of $k$ convex polyhedra*, in Computational Geometry and Its Applications, Lecture Notes in Comput. Sci. 333, Springer-Verlag, New York, 1988, pp. 180–186.

[56] T. Roos and P. Widmayer, *$k$-violation linear programming*, Inform. Process Lett., 52 (1994), pp. 109–114.

[57] R. Seidel, *Small-dimensional linear programming and convex hulls made easy*, Discrete Comput. Geom., 6 (1991), pp. 423–434.

[58] M. Sharir, *On $k$-sets in arrangements of curves and surfaces*, Discrete Comput. Geom., 6 (1991), pp. 593–613.

[59] M. Sharir, S. Smorodinsky, and G. Tardos, *An improved bound for $k$-sets in three dimensions*, Discrete Comput. Geom., 26 (2001), pp. 195–204.

[60] M. Sharir and E. Welzl, *A combinatorial bound for linear programming and related problems*, in Proceedings of the 9th Symposium on Theoretical Aspects of Computer Science, Lecture Notes in Comput. Sci. 577, Springer-Verlag, New York, 1992, pp. 569–579.

SIAM J. COMPUT.
Vol. 34, No. 4, pp. 894–923

# DYNAMIC LCA QUERIES ON TREES*

RICHARD COLE† AND RAMESH HARIHARAN‡

**Abstract.** We show how to maintain a data structure on trees which allows for the following operations, all in worst-case constant time:

1. insertion of leaves and internal nodes,
2. deletion of leaves,
3. deletion of internal nodes with only one child,
4. determining the least common ancestor of any two nodes.

We also generalize the Dietz–Sleator "cup-filling" scheduling methodology, which may be of independent interest.

**Key words.** LCA, dynamic LCA, "cup-filling" scheduling

**AMS subject classifications.** 68W05, 68W40, 68Q25, 68P05

**DOI.** 10.1137/S0097539700370539

## 1. Introduction.

Finding *least common ancestors (LCAs)* in trees is a fundamental problem that arises in a number of applications. For example, it arises in computing maximum weight matchings in graphs [Ga90], in computing longest common extensions of strings, finding maximal palindromes in strings, matching patterns with $k$ mismatches, and finding $k$-mismatch tandem repeats [Gus97]. The tree involved in all but the first of these applications is a *suffix tree*.

The primary use of LCA computations in a suffix tree is to determine the longest common prefix of two substrings in constant time. This operation is used heavily in the above applications. The suffix tree for a given string can be constructed in linear time [M76]. Each node in this tree corresponds to a substring of the given string. The longest common prefix of any two substrings is the string corresponding to the LCA of the corresponding nodes.

The first constant time LCA computation algorithm was developed by Harel and Tarjan [HT84]. This algorithm preprocesses a tree in linear time and subsequently answers LCA queries in constant time. Subsequently, Schieber and Vishkin [SV88], Berkman and Vishkin [BV94], and Bender and Farach-Colton [BF00], gave simpler algorithms with the same performance.

In this paper, we consider the dynamic version of the problem, i.e., maintaining a data structure which supports the following tree operations: insertion of leaves and internal nodes, deletion of internal nodes with only one child, and LCA queries. We assume that when a new node is inserted, a pointer to the insertion site in the tree is also given. The motivation is to maintain a suffix tree under insertion of new strings, deletion of strings, and longest common prefix queries. One application of this problem arises in maintaining a databaseof strings in order to answer queries of the following kind: given a pattern string, find all its occurrences with up to $k$ mismatches in the

*Received by the editors April 5, 2000; accepted for publication (in revised form) July 6, 2004; published electronically May 12, 2005. This work was supported by NSF grants CCR9503309, CCR9800085, and CCR0105678.

http://www.siam.org/journals/sicomp/34-4/37053.html

†Department of Computer Science, Courant Institute of Mathematical Sciences, New York University, 251 Mercer Street, New York, NY 10012 (cole@cs.nyu.edu).

‡Department of Computer Science and Automation, Indian Institute of Science, Bangalore 560012, India (ramesh@csa.iisc.ernet.in). This work was done in part while the author was visiting NYU.

strings in the database. Efficient algorithms for finding all occurrences of a pattern in a text with up to $k$ mismatches [LV86, CH97] require maintaining the suffix tree of the text and processing it for LCA queries. Extending these algorithms to maintain a database of strings supporting $k$-mismatch queries would require maintaining LCAs dynamically.[1] Additions and deletions of new strings to the database will change the suffix tree as well as the LCA data structure. A new string can be inserted into a suffix tree in time proportional to its length. The number of nodes inserted in the process is proportional to the length of the string inserted. These nodes could be leaves or internal nodes. Similarly, deletion of a string will cause the removal of some nodes. Our goal is to minimize the work needed to maintain the data structure for each node inserted or deleted.

Harel and Tarjan [HT84] gave an algorithm to maintain a forest under *linking* and finding LCAs. This is useful in computing maximum weight matchings in graphs. The link operation generalizes insertions of new leaves. Harel and Tarjan's link operation allowed only linking of whole trees, not linking of a tree to a *subtree* of another tree. The amortized time taken by their link operation was $\alpha(m,n)$, where $n$ is the size of the tree and $m$ is the number of operations. LCA queries were answered in constant time. Gabow [Ga90] gave an algorithm which performs additions and deletions of leaves in constant amortized time and also supports linking of trees to subtrees in $\alpha(m,n)$ amortized time. The worst-case time for update operations in both these algorithms was $\Omega(n)$. The worst-case time for an LCA query was $O(1)$. Both the above algorithms were motivated by the maximum weighted matching problem in graphs.

Since our focus is different, namely suffix trees, we consider insertions and deletions of leaves and internal nodes, but not the link operation. Note that neither of the above algorithms considered insertions of internal nodes. Westbrook [We92] built upon Gabow's approach above to give an $O(1)$ amortized time algorithm which could perform insertions and deletions of leaves as well as internal nodes. Our focus, however, is on worst-case insertion time rather than amortized time.

We give an algorithm which performs insertions and deletions of leaves and internal nodes while supporting LCA queries, all in *constant worst-case* time. This algorithm is obtained in two stages. First, we give an algorithm which takes $O(\log^3 n)$ worst-case time for insertions and deletions and $O(1)$ worst-case time for queries. This is the core of our algorithm. Subsequently, we show how to improve the worst-case time for insertions and deletions to $O(1)$ by using a standard multilevel scheme. The space taken by our algorithm is $O(n)$.

Our basic $O(\log^3 n)$ worst-case time algorithm broadly follows Gabow's and Schieber and Vishkin's algorithm. The overall approach is to decompose the tree into centroid paths and assign a code to each node. From the codes for two given nodes, the centroid path at which their paths from the root separate can be easily determined in constant time. And given two vertices on the same centroid path, the one closer to the root can be determined by a simple numbering. Together, the codes and numberings yield the LCA. The basic problem we have to solve is to maintain the centroid paths and codes over insertions and deletions. Gabow's algorithm does this in bursts, reorganizing whole subtrees when they grow too large. This makes the worst-case time large. However, the amortized time is $O(\log n)$ because each reorganization is coupled

[1]However, just maintaining LCAs alone is not sufficient to solve the dynamic $k$ mismatches problem with query time smaller than the obvious static algorithm. Therefore, we do not obtain any results on the dynamic $k$ mismatches problem here.

with a doubling in size; this time is reduced to $O(1)$ using a multilevel scheme. Note, also, that Gabow does not consider insertions of internal nodes. Thus, two main issues need to be tackled to get constant worst-case time.

The first issue is that of maintaining numbers on centroid paths so that the LCA of two given nodes on the same centroid path can be found in constant time. For this purpose, we use the Dietz–Sleator [DS87] data structure (or the Tsakalidis [Ts84] data structure) which maintains order in a list under insertions and deletions.[2]

The second and the more serious issue by far is that of reorganizing trees to maintain centroid paths and codes in constant worst-case time. Since we seek constant worst-case time, there is little option but to delay this reorganization. We amortize this reorganization over future insertions and deletions, i.e., spread the $O(1)$ amortized work of Gabow's algorithm over future insertions/deletions so each insertion and deletion does only a constant amount of work. This approach is not new and has been used by Dietz and Sleator [DS87] and Willard [W82] among others. However, the problems caused by this approach are nontrivial and specific to this setting.

The problem with this approach is that any change at a node $v$ causes the codes at all the nodes in the subtree rooted at $v$ to change. Since updates of these codes are spread over future insertions and deletions, queries at any given instant will find a mixture of updated and not yet updated codes. This could potentially give wrong answers. We avoid this with a two-phase update of a two-copy code.

What further complicates the situation is that reorganizations could be taking place at many subtrees simultaneously, one nested inside the other. This implies that the variation amongst nodes in the degree to which their codes have been updated at any given instant could be arbitrarily large. Nonetheless, the two-phase update ensures correct answers to the queries.

An additional complication is that the various nested reorganizations could proceed at very different speeds, depending upon the distribution of the inserted nodes. In this respect, the situation is analogous to that encountered in asynchronous distributed computing, where interacting processes proceeding at arbitrarily different speeds need to ensure they collectively make progress on their shared computation.

Our main contribution is to organize the various nested processes so that they complete in time and also maintain codes which give correct answers to queries. This is obtained by a nontrivial scheduling procedure coupled with an analysis which bounds the total sizes of nested processes.

To understand our scheduling procedure it is helpful to recall the Dietz–Sleator cup-filling methodology. It concerns a collection of tasks in which priorities increase in an unknown but bounded way (i.e., adversarially set), each time unit; the scheduling is very simple: simply select the current highest priority task and run it to completion. They show this algorithm has a good performance which they tightly bound. We are concerned with a similar scenario, but in which priorities are only approximately known; naturally, we schedule the apparently highest priority task. We also allow the tasks to be somewhat divisible so that they need not be run to completion once started. Details appear in section 6.8.

## 2. Overview.

We assume that the tree is a binary tree, without loss of generality.

[2] We can also use the data structure given here but supporting only leaf insertion and deletion. This results in the centroid paths being modified only at their endpoints, and a trivial numbering scheme suffices to maintain order. This approach was suggested by Farach-Colton [F99].

First, we consider only insertions. Deletions are handled easily by just ignoring them until they form a significant fraction of the number of nodes, at which point the entire data structure is rebuilt. The original data structure is also maintained until this rebuilding is complete in order to answer queries. Details on handling deletions are deferred to section 8.

We also assume that the insertions at most double the tree size. This assumption is also handled easily by rebuilding when the size of the tree increases by some suitable constant factor, and again is addressed in section 8.

The paper is organized as follows. We give some definitions in section 3. Then we describe the algorithm for the static case in section 4 and Gabow's dynamic algorithm in section 5. In section 6, we describe our $O(\log^3 n)$ worst-case time algorithm. Sections 7 and 8 describe the improvement to $O(1)$ time and the handling of deletions, respectively.

**3. Definitions.** We partition the tree $T$ into paths, called *centroid paths*, as follows. Let $T_y$ denote the subtree of $T$ rooted at node $y$. Suppose $2^i \leq |T_y| < 2^{i+1}$. Then, $y$ is called a *tail* node if $|T_z| < 2^i$ for all children $z$ of $y$, if any. Such vertices $y$ will lie in distinct centroid paths and will be the *tails*, i.e., bottommost vertices, in their respective centroid paths. The centroid path containing $y$ connects $y$ to its farthest ancestor $x$ such that $2^i \leq |T_x| < 2^{i+1}$. $x$ is called the *head* of this path. It is easy to see that centroid paths defined as above are disjoint.

A centroid path $\pi$ is said to be an ancestor of a node $x$ if $\pi$ contains an ancestor[3] of $x$. A centroid path $\pi$ is said to be an ancestor of another path $\pi'$ if $\pi$ is an ancestor of the head of $\pi'$. A centroid path $\pi$ is a child of another path $\pi'$ if the head of $\pi$ is a child of a node on $\pi'$.

The *least common centroid path (LCCP)* of two nodes is the centroid path containing their LCA. An *off-path node* with respect to a particular centroid path $\pi$ is a node not on $\pi$ whose parent is on $\pi$. The *branching pair (BP)* of two nodes $x, y$ is the pair of nodes $x', y'$ on the LCCP which are the least common ancestors of $x, y$, respectively.

**4. Outline of the static algorithm.** The nodes of the tree are partitioned into centroid paths. The nodes are then numbered so that parents have smaller numbers than their children. In fact, the numbering need satisfy only the following property: if $x$ and $y$ are distinct vertices on the same centroid path and $x$ is a strict ancestor of $y$ then $number(x) < number(y)$.

Each vertex is given a code of length $O(\log n)$ with the following property: the LCCP and BP of $x$ and $y$ can be determined easily from the first bit in which the codes for $x$ and $y$ differ. Let $code(x)$ denote the code for node $x$.

The LCA of two nodes $x, y$ is now easy to determine. The LCCP and BP of $x, y$ are found in constant time using a RAM operation for finding the leftmost bit which differs in $code(x)$ and $code(y)$.[4] Note that the nodes in the BP need not be distinct (see Figure 1). The node in the BP with the smaller number is the desired LCA.

**4.1. The codes.** We still need to describe the assignment of codes to nodes. Note that if the tree was a complete binary tree, all centroid paths would be just single nodes. Furthermore, $code(x)$ could be the canonical code obtained by labelling the left-going edges 0 and right-going edges 1, and reading off the path labels from the root to $x$.

[3] All references to ancestors in this paper will be in the nonstrict sense, unless otherwise stated.
[4] Or perhaps, using table look-up on a precomputed set of answers.

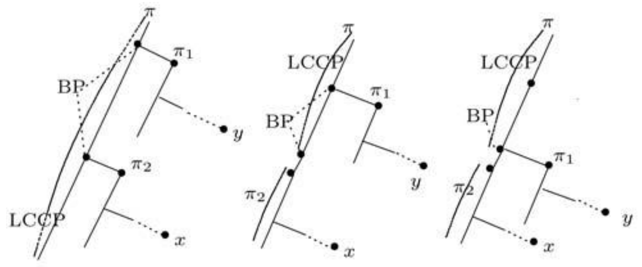


FIG. 1. *LCCP and BP.*

For a general tree, *code*$(x)$ is a concatenation of smaller bit strings, one for each centroid path containing an ancestor of $x$.

First, we assign to each centroid path $\pi$ a bit string called *separator*$(\pi)$. These strings have the following property. For each centroid path $\pi$, the separator strings assigned to children centroid paths of $\pi$ form a prefix-free set (i.e., no string is a prefix of another string). The length of *separator*$(\pi)$ is $O(\log \frac{|T_x|}{|T_y|})$, where $y$ is the head of $\pi$ and $x$ is the head of the centroid path containing the parent of $y$.

*code*$(x)$ is a concatenation of the separator strings assigned to ancestor centroid paths of $x$ (including the path containing $x$) in order of increasing distance from the root. It is easy to show that the length of the code is $O(\log n)$ (take any sequence of centroid paths encountered on a path from the root to a leaf and let $x_1 \ldots x_k$ be the heads of these centroid paths; then the sum $\sum_{i=2}^{k} \log \frac{|T_{x_{i-1}}|}{|T_{x_i}|}$ equals $O(\log |T_{x_1}|) = O(\log n)$).

It will be convenient to have *separator*$(\pi)$ be of length $a(\lceil \log |T_x| \rceil - \lceil \log |T_y| \rceil)$ for a suitable constant integer $a \geq 1$, if need be by padding *separator*$(\pi)$ with zeros at the right end. This ensures that the length and position of *separator*$(\pi)$ in a code is fully determined by $|T_x|$ and $|T_y|$.

Each separator string in *code*$(x)$ is *tagged* with the name of the corresponding centroid path, i.e., given the index of a bit in *code*$(x)$, we can recover the name of the path within whose separator this bit lies, in $O(1)$ time. The functions *number*$(x)$, *separator*$(\pi)$, *code*$(x)$, and the above tagging can all be computed in $O(n)$ time (we comment briefly on this below).

The LCCP of nodes $x$ and $y$ is determined from *code*$(x)$ and *code*$(y)$ in $O(1)$ time as follows. We find the leftmost bit in which *code*$(x)$ and *code*$(y)$ differ; subsequently, using the above tagging, we find the name of the two paths whose separators contain this mismatch bit in *code*$(x)$ and *code*$(y)$, respectively. The parents of the heads of these two paths will give the BP (see Figure 1) and the path containing this BP is the LCCP. To see this, note that the separator strings in both codes corresponding to the LCCP and centroid paths above the LCCP are identical. In addition, due to the above prefix-free property, the separator strings corresponding to the two children paths of the LCCP which are ancestors of $x$ and $y$, respectively, necessarily differ in some bit.

A special case arises when one or both of $x, y$ are part of the LCCP. If both are part of the LCCP, then the one with smaller *number*() is the LCA. Otherwise, if $x$ is part of the LCCP but $y$ is not, then *code*$(x)$ is a prefix of *code*$(y)$. The path containing $x$ is the LCCP; BP is easy to determine as well.

*Computation.* We briefly touch upon how *number*$(x)$, *separator*$(\pi)$, and *code*$(x)$

can be computed in $O(n)$ time and, further, how each separator string in *code*$(x)$ can be tagged with the name of the corresponding path.

Computing *number*$(x)$ is clearly easy: in any centroid path the numbers only need to be in increasing order of distance from the root.

Computing *separator*$(\pi)$ involves assigning prefix-free codes. We outline how this is done for child paths of a centroid path $\pi$ with head $x$, given that the separator for $\pi$ has already been determined. Let $\pi_1 \dots \pi_k$ denote the child paths of $\pi$ and $x_1 \dots x_k$ their respective heads. We construct a weight-balanced binary search tree on the weights $|T_{x_1}| \dots |T_{x_k}|$. This tree can be constructed in $O(k)$ time [Meh77] and has the property that the height of the leaf corresponding to $x_i$ is $O(\log \frac{\sum_{j=1}^{k} |T_{x_j}|}{|T_{x_i}|}) = O(\log \frac{|T_x|}{|T_{x_i}|})$. Separator codes for $\pi_1 \dots \pi_k$ are obtained by encoding left edges in the weight-balanced tree by 0, encoding right edges by 1, and reading off the labels on the path from the root to the appropriate leaves in this tree. Clearly, codes thus obtained are prefix-free. The whole procedure takes $O(k)$ time, which translates to $O(n)$ time over all of $T$.

*code*$(x)$ is computed in $O(1)$ time from *code*$(y)$, where $y$ is the parent of $x$, as follows. If $x$ and $y$ are in the same centroid path, then the codes are the same. Otherwise, $x$ is in a child path $\pi$ of the path containing $y$, and *code*$(x)$ is obtained by concatenating *code*$(y)$ and *separator*$(\pi)$. This is done in $O(1)$ time using a RAM operation.

There is one issue which needs clarification. Recall the tagging mentioned above. One method to find the name of the centroid path whose separator string contains a particular mismatch bit is to keep an array of size $O(\log n)$ for each vertex $x$; the array for vertex $x$ stores the relevant path name for each potential mismatch bit. Clearly, given the leftmost bit in which *code*$(x)$ differs from *code*$(y)$, indexing into the above arrays (one each for $x$ and $y$) using the location of the mismatch bit will give us the names of the required separator paths in $O(1)$ time. However, setting this up would require $O(n \log n)$ space and, therefore, $O(n \log n)$ time, over all nodes $x$. Both terms can be reduced to $O(n)$ in one of two ways.

The first involves using a multilevel data structure, similar to the one used by Gabow [Ga90] and the one we use to get $O(1)$ query time for the dynamic case; this is elaborated upon further in section 7. In this paper, we will assume the framework of this solution.

In the second solution, this tagging is avoided altogether. Instead, centroid paths are named by the code given to their respective heads and the name of the LCCP of two given nodes $x$ and $y$ is easily recovered in $O(1)$ time, given their codes. Indeed, only the following operation needs to be performed to determine the name of the LCCP: given the mismatch bit in *code*$(x)$, return the prefix of *code*$(x)$ comprising separators of all those centroid paths which are ancestors of that centroid path whose separator contains the mismatch bit (and likewise for *code*$(y)$). This is easily done using look-up tables of $O(n)$ size.

## 5. The dynamic case: Gabow's amortized bound.

The main problem in the dynamic case is to maintain the centroid paths along with the quantities *number*$(x)$, *separator*$(\pi)$, and *code*$(x)$.

Gabow [Ga90] gave an algorithm for the case when only leaf insertions were allowed. Maintenance of *number*$(x)$ is trivial in this case: new leaves are assigned successively larger numbers. However, if insertions of internal nodes is allowed, then it is not clear how to maintain *number*$(x)$.

Gabow's approach to maintaining centroid paths is as follows. As insertions are made, the centroid paths in the tree will change, in a manner yet to be described. Gabow updates the centroid paths not incrementally but in bursts. Whenever the subtree rooted at the head of a centroid path doubles[5] in size, the entire subtree is reorganized, i.e., reprocessed to construct new centroid paths, separators, and codes.

Gabow maintains separators and codes as follows. Instead of prefix-free separators, Gabow maintains a family of nested intervals. The interval for a centroid path is a subinterval of the interval for any ancestor centroid path. In addition, the intervals for the centroid paths which are children of a path $\pi$ are all disjoint. A constrained version of this approach is equivalent to maintaining separators, as we shall describe shortly in section 6.2.

When a new off-path node $y$ with respect to a particular centroid path $\pi$ is inserted, a new interval within the interval for $\pi$ and to the right of all other intervals for children of $\pi$ is assigned to $y$. Gabow shows that there is always sufficient space for this new interval, given that a subtree is reprocessed whenever its size doubles, at which point intervals nested within another are packed together. We follow a similar approach.

The time taken by Gabow's algorithm on any single insertion is proportional to the size of the subtree which is reorganized. Thus the worst-case time for an insertion could be $\Omega(n)$. However, since the reorganization of a subtree is coupled with the doubling in its size, the amortized time for an insertion is $O(\log n)$. Gabow converts this to $O(1)$ amortized time by using a multilevel approach.

## 6. Our $O(\log^3 n)$ time worst-case algorithm.

As described above, there are two main hurdles to improving Gabow's scheme to run in constant worst-case time, or even poly-logarithmic worst-case time. The first is the maintenance of *number*$(x)$ when internal nodes are inserted. The second is the reorganization of subtrees.

The first problem is easy to overcome using an algorithm for maintaining order in a list under insertions and deletions in $O(1)$ worst-case time, due to Dietz and Sleator [DS87]. We maintain each centroid path as an ordered list using this algorithm, allowing us to answer queries about which node in a particular BP is closer to the root in $O(1)$ worst-case time.

The second problem is more serious. Our basic approach is to distribute the reorganization of a subtree caused by a particular insertion over subsequent insertions. In other words, the various operations involved in reorganizing a subtree are performed, a poly-logarithmic number at a time, over future insertions.[6] This means that queries which come while a subtree is being reorganized will see a partially reorganized subtree and therefore risk returning wrong answers. We describe our algorithm for the reorganization in further detail next.

### 6.1. Weighted nodes.

Our $O(1)$ time algorithm for the LCA problem uses as a subroutine an $O(\log^3 n)$ algorithm for a slightly generalized problem. We indicate the reasons behind the need for a generalization next.

Let $T$ be the tree on which the LCA queries are being performed. Our approach is to select $\Theta(n/\log^3 n)$ nodes of $T$, which partition $T$ into subtrees of size $O(\log^3 n)$, called *small* subtrees. The selected nodes are formed into an induced tree $T_1$, to which we apply the $O(\log^3 n)$ time algorithm. It will be the case that for each $\Theta(\log^3 n)$

[5] When it crosses a power of two boundary, actually.

[6] This general approach has also been followed by Dietz and Sleator [DS87] and by Willard [W82] to convert algorithms with good amortized performance to worst-case performance.

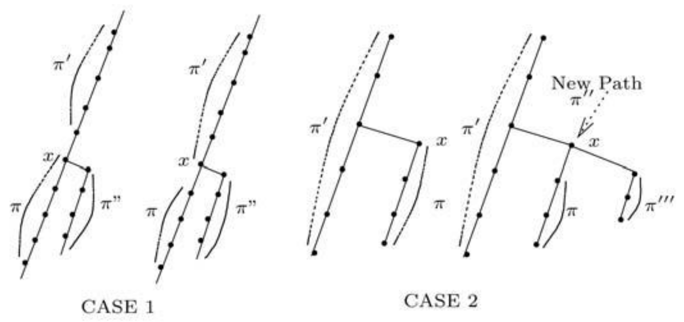


FIG. 2. *Changing centroid paths.*

insertions into one of these small subtrees just $O(1)$ nodes are added to $T_1$. To achieve the $O(1)$ worst-case time bound, we need to perform the $O(\log^3 n)$ operations stemming from an insertion to $T_1$ over the corresponding $O(\log^3 n)$ insertions to the relevant small subtree of $T$, at a rate of $O(1)$ operations per insertion. To control this appropriately, we weight the nodes of $T_1$ as follows.

Weight constraints:

(i) All weights are integer multiples of $1/\log^3 n$.[7]

(ii) Node weights are in the range $[0, 1]$.

(iii) If a node has weight less than 1, its parent has weight 1.

(iv) A weight 1 node has at most one child of weight less than 1.

Weight increases occur in integer multiples of $1/\log^3 n$; the largest possible increase is by $1/\log^2 n$, as we will see in Remark 6.23. We will show that we can maintain $T_1$ with $O(\log^3 n)$ operations per unit increase in weight. Later we will see that each insertion to $T$ results in at most a $4/\log^3 n$ increase in weight, and we will show that $T_1$ can be maintained with $O(1)$ work per insertion to $T$. For intuition, the reader may find it helpful to think of nodes being inserted with weight 1 with the caveat that this is not exactly the scenario we are describing.

When a node is inserted in $T_1$ it will have weight zero initially. As the relevant inservations to $T$ occur, its weight is increased. Until its weight reaches 1, no further nodes can be inserted in its neighborhood in $T_1$, so as to meet constraints (iii) and (iv) above.

**6.2. Updating centroid paths.** When a node's weight is increased (by up to $1/\log^2 n$), each of its $O(\log n)$ ancestor centroid paths could shift down by one or two nodes as shown in Figure 2, Case 1. New centroid paths of one or two nodes could begin as well, as shown in Figure 2, Case 2. (If all node weights are equal to 1, the shifts are only by one node and the new paths each have only one node.)

We would like to maintain the invariant that for each centroid path there is an integer $i$ such that for each node $w$ on the path, $2^i \le |T_w| < 2^{i+1}$, where $|T_w|$ denotes the weight of the subtree $T_w$ rooted at $w$. We call such paths *$i$-paths.* Unfortunately, as it is expensive to update the codes associated with the subtrees of a centroid path, the changes to a centroid path may lag the increases in the sizes of the trees $T_w$. Consequently, we will allow centroid paths to overlap.

More specifically, an $i$-path $\pi$, as in the static structure, comprises a maximal

[7]$n$ is restricted to a range $[2^i, 2^{i+a}]$ for some constant $a$, and we take $\log^3 n$ to be the fixed value $(i+a)^3$.

sequence of nodes $w$ with $2^i \leq |T_w| < 2^{i+1}$, but in addition may include further nodes $z$ with $2^{i+1} \leq |T_z| < 2^{i+1} + 2^{i-1} + 2^{i-2}$, for $i \geq 1$. Any such node $z$ is also part of an $(i+1)$-path $\pi'$. Naturally, the nodes of $\pi$ are required to form a path.

Further, to accommodate leaves, which may have weight 0, we define a 0-path to comprise a maximal sequence of nodes $w$ such that $0 \leq |T_w| < 2$, and in addition it may include further nodes $z$ with $2 \leq |T_z| < 2\,\frac{3}{4}$.

If node $v$ lies on both an $i$-path $\pi$ and an $(i+1)$-path $\pi'$, $\pi$ is said to be its *primary* path and $\pi'$ its *secondary* path. If $v$ lies on a single path $\pi$, $\pi$ is its primary path.

We need to redefine the notion of parenthood for these paths.

DEFINITION 6.1. *Let $\pi$ be a centroid path with topmost node $x$, called head($\pi$). If $x$ is secondary on path $\pi'$, then $\pi'$ is the parent of $\pi$. Otherwise, if $x$ is not secondary on any path, the parent of $\pi$ is given by the primary path of parent($x$).*

LEMMA 6.2. *Suppose that $u$ and $v$ are adjacent secondary nodes on $(i+1)$-path $\pi'$. Then $u$ and $v$ are primary nodes on the same $i$-path $\pi$, where $\pi'$ is the parent of $\pi$.*

*Proof.* W.L.O.G., let $u$ be the parent of $v$. $|T_v| \geq 2^{i+1}$ as $v$ lies on $\pi'$. Let $w$ be $u$'s other child, if any. Suppose, for a contradiction, that $u$ and $v$ were on different $i$-paths. Also suppose that $u$ is on $i$-path $\pi$. Then $w$ must have been part of $\pi$ before $|T_v|$ reached $2^i$, as otherwise at that point $v$ would have joined $\pi$. For $i \geq 1$, it follows that $|T_w| \geq 2^i$. For $i = 0$, by weight constraint (iv), $weight(w)$ must reach 1 before $v$ is inserted (with initial weight 0) in $T_1$; thus here too, $|T_w| \geq 2^i$. Thus $|T_u| = weight(u) + |T_w| + |T_v| \geq 2^{i+1} + 2^i$, and thus $u$ cannot be on an $i$-path, contrary to assumption. ∎

The increment in weight of a node $z$ may cause changes to the tails of some or all of its ancestral centroid paths. Changes to the heads of the paths may be deferred; their handling is described in subsequent sections. The reason for delaying changes to the head is that such a change entails updating the codes of all the nodes in the off-path subtree of the head node. The following changes may occur to an $i$-path with tail $y$ and head $x$.

1. If node $z$ is not a descendant of $y$, then the tail of $\pi$ is unchanged.
2. If $z$ is a descendant of $x$ and $|T_x|$ increases from less than $2^{i+1}$ to at least $2^{i+1}$ due to the weight change, then $x$ is added to an $(i+1)$-path. If the path $\pi''$, the parent of $\pi$, is an $(i+1)$-path, then $x$ becomes the tail of $\pi''$. If not, a new $(i+1)$-path is created, comprising the single node $x$. In any event, $x$ remains on $\pi$ for now. Note that there may be two nodes $x_1$ and $x_2$ for which $|T_{x_h}|$, $h = 1, 2$, increases from less than $2^{i+1}$ to at least $2^{i+1}$.

REMARK 6.3. *Clearly, as weights increase in the subtree rooted at the head $x$ of path $\pi$, eventually $x$ must be removed from the head of $\pi$ in order to maintain the invariant that $|T_{head(\pi)}| < 2^{i+1} + 2^{i-1} + 2^{i-2}$. Thus, over time, the path $\pi$ will migrate down the (growing) tree, but will never disappear.*

REMARK 6.4. *Actually, the tail node $u$ of an $i$-path might have children $v$ and $w$ with $|T_v|, |T_w| < 2^{i-1}$, but at least one of $|T_v|, |T_w|$ will be of size $2^{i-2}$ or larger (for to contradict this we would need $2^i \leq |T_u| = wt(u) + |T_v| + |T_w| < 1 + 2 \cdot 2^{i-2}$, i.e., $2^{i-1} < 1$ or $i < 1$, and then $u$ is a leaf). Thus as $(i-1)$-path $\pi'$ migrates down the tree from such a node $u$ it might disappear; to avoid this we preserve it as a zero length path at the "bottom" of node $u$ and ancestral to both $v$ and $w$. Later if either $|T_v|$ or $|T_w|$ reaches size $2^{i-1}$, then the corresponding node ($v$ or $w$) joins $\pi'$. When a node $z$ is inserted it joins a centroid path according to the following rules.*

1. If $z$ is inserted between two nodes in $\pi$, then $z$ is added to $\pi$ at the appropriate place (possibly, $z$ is added to two paths in this way, once as a primary node and once as a secondary node).

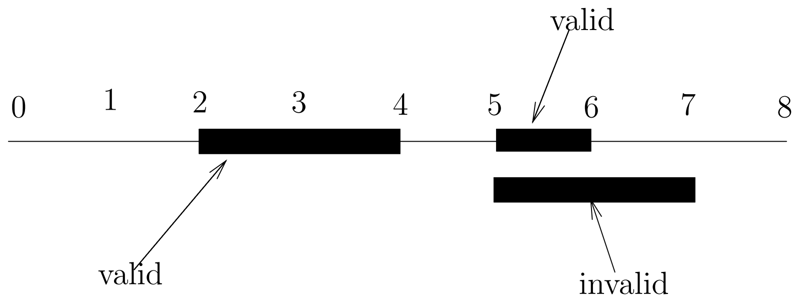


FIG. 3. *Constrained intervals.*

2. If node $z$ is inserted between $x$, the head of path $\pi$, and $x$'s parent $z'$, then $z$ is added to the centroid path or paths to which $x$ belongs. If as a result $z$ is on both an $i$- and an $(i+1)$-path, it will be the case that $|T_z| < 2^{i+1}+2^{i-1}+2^{i-2}$, since $|T_z| = |T_x|$ at this point.

**6.3. Updating separators.** Separators and codes have to be updated to reflect the above changes in the centroid paths; the update begins when a node previously on an $i$-path joins an $(i+1)$-path. When the update completes, the node leaves the $i$-path. In between times it lies on both paths.

The following interpretation of separators in terms of intervals is helpful.

*Separators as constrained intervals.* Consider an interval of length $m = 2^k$. All subintervals we consider will have lengths which are powers of 2. Each integer point on this interval has an associated $k$ bit code (the leftmost point is coded with the all 0s string, subsequent points are coded by the binary encoding of their distance from the leftmost point; this encoding has length $k$). We allow a subinterval of length $2^i$ to begin only at those integer points which have $i$ trailing 0s in their bits (see Figure 3); with such a subinterval, we associate a bit string of length $k-i$ given by the leading $k-i$ bits of the code for the starting point. It can easily be seen that given a set of disjoint subintervals with this property, the bit strings assigned to the subintervals form a prefix-free set. Thus assigning prefix-free separators is identical to assigning subintervals with the above constraints. Henceforth, all our references to intervals will be to intervals with the above constraints.

*Mapping paths to intervals.* With each $i$-path $\pi$ we maintain an interval $int_\pi$ of length either $2^{ic}$ or $2^{ic+c'}$, $c' < c$, where $c$ and $c'$ are constants to be determined. When $\pi$ is created, $int_\pi$ has length $2^{ic}$. At some point between the time $T_{head(\pi)}$ reaches size $2^{i+1} - 2^{i-3}$ and the time it reaches size $2^{i+1}$, $int_\pi$ will gain length $2^{ic+c'}$. There are two goals. The first is to ensure that if the parent of $head(\pi)$ lies on an $i$-path (not $\pi$) as well as an $(i+1)$-path, then $int_\pi$ has length $2^{ic}$. The second is to ensure that if a node originally on $\pi$ is also secondary on path $\pi'$ then $int_\pi$ has length $2^{ic+c'}$. By definition, once $T_{head(\pi)}$ first reaches size $2^i+2^{i-1}+2^{i-2}$ the first situation no longer applies. The second situation applies once $T_{head(\pi)}$ reaches size $2^{i+1}$ (as $head(\pi)$ changes the size of $T_{head(\pi)}$ may subsequently drop). Note that constraints on $c$ and $c'$ will be imposed by Lemma 6.10 below; setting $c' = 5$ and $c = 10$ suffices.

A crucial property of an interval for an $i$-path $\pi$ is that the rightmost bit for its separator ends at a fixed location, so that in each code in which it occurs there are a specified number of bits to its right whose values depend only on the separators for $h$-paths, $h < i$, to which the relevant node belongs. The number of these bits is either $ic$ or $ic + c'$, corresponding to the size of $int_\pi$. This allows the code for $int_\pi$

to be changed without requiring any change to the separators for $h$-paths, $h < i$, contained in the codes in which $int_\pi$ occurs. The one exception arises when the size of the interval for $\pi$ increases. But this can be viewed as simply prefixing $c'$ zeros to the separator following $\pi$ in the code for each node in $T_{head(\pi)}$.

*Updating intervals.* The following updates need to be performed. See Figure 2.

1. Node $x$ lies on paths $\pi$ and $\pi'$; $x$ is the head of $\pi$ and is being removed from $\pi$; in addition, there is a proper ancestor of $x$ that is or had been on $\pi'$. Then the centroid path $\pi''$ whose head was the off-path child of $x$ (with respect to $\pi$) must be assigned a new interval. The interval for $\pi''$ was earlier nested within $int_\pi$ and $int_{\pi'}$. Now this interval must be reassigned so as to be disjoint from $int_\pi$ but still nested within $int_{\pi'}$. The process which does this is called *Reassign*$(x)$.
2. Node $x$ is on paths $\pi$ and $\pi'$, and $x$ is the head node of $\pi$ and $\pi'$. Then $\pi'$ is a new centroid path, and a new interval has to be assigned to $\pi'$. This interval $int_{\pi'}$ must be nested within the interval $int_{\pi''}$ for the path $\pi''$, the previous parent of $\pi$, and now the parent of $\pi'$. Further, $int_\pi$ must be reassigned so it is nested within $int_{\pi'}$. This is done by a procedure called *Assign*$(\pi')$. In addition, the interval associated with the path $\pi''$ containing the off-path child of $x$ (with respect to $\pi$) must be reassigned so that it is also nested within $int_{\pi'}$. This is done by a procedure called *Reassign*$(x)$. There are a few details associated with this case which will be explained later in section 6.4.
3. Node $x$ is the head of $i$-path $\pi$, and its parent $y$ lies on class $i$ path $\pi' \neq \pi$ and also necessarily on class $i+1$ path $\pi''$ (for by the maximality of $\pi$, $|T_y| \geq 2^{i+1}$). Note that by the time $|T_{head(\pi)}| = 2^{i+1}$, $y$ will no longer be on $\pi'$. Between this time and before the time $|T_{head(\pi)}|$ reaches $2^{i+1} + 2^{i-1} + 2^{i-2}$, a new larger interval will have been assigned to path $\pi$; this new interval will be contained within $int_{\pi''}$. This is done by a process called *Rescale*$(\pi)$.

Thus as time proceeds, subintervals move from one interval to another and new intervals are created. This movement and creation is done as follows. When a subinterval has to be removed from an interval, the subinterval is just marked as deleted but not removed. When a new subinterval has to be assigned to $\pi$ within $int_{\pi'}$, where $\pi'$ is the parent path of $\pi$, it is assigned in either the first half of $int_{\pi'}$ or the second half of $int_{\pi'}$, based on a logic to be described. In either case, it is assigned to the leftmost available slot to the right of all assigned subintervals in constant time. Note that a particular weight increase creates at most constant number of *Rescale*, *Reassign*, or *Assign* processes at each ancestor centroid path of the reweighted node.

We need to ensure that *Assign*$(\pi)$, *Rescale*$(\pi)$, and *Reassign*$(x)$ will always find an empty slot as above to assign to the new interval for $\pi$. This is not hard to ensure if nondeleted subintervals are separated by small gaps only. However, large gaps could build up as subintervals enter and leave intervals. Consequently, we need a background process which will start removing deleted subintervals and compacting nondeleted subintervals within an interval, once the interval reaches a certain fraction of its capacity. This process is described next.

*The compacting process for an $i$-path $\pi$.* The compacting process maintains the interval $int_\pi$ as two halves. At any given time, one half, the insertion half, will be receiving newly inserted subintervals. At the same time the deleted subintervals in the other half are being removed and the nondeleted subintervals are being moved to the insertion half. By the time the insertion half is filled, the noninsertion half will be empty, and then their roles are toggled. Actually, the toggling occurs at a predetermined time when it is guaranteed that the noninsertion half is empty and the insertion half has not overflowed. W.L.O.G., consider the instant at which the right half becomes the insertion half. The compaction process moves through the subintervals in the left half from left to right removing deleted subintervals and reassigning undeleted subintervals at the leftmost possible slot in the right half (to the right of already assigned subintervals in this half). Insertions subsequent to the beginning of the compaction process will also be made in the right half until the stopping time, which is no later than when the right half becomes full. We will show that the compaction process in the left half will have finished by the time this happens. At this point, the compaction process will become active in the right half and insertions will be performed in the left half. We call the above process *Compact*$(\pi)$. Note that a single insertion can initiate a compaction process at each of its ancestor centroid paths.

Thus there are four kinds of processes which could be created when a node is inserted: *Assign*( ), *Rescale*( ), *Reassign*( ), and *Compact*( ). Each process is expensive and takes time at least proportioned to the size of the subtree in which it modifies codes (this updating of codes will be elaborated upon shortly in section 6.4). Thus these processes have to be performed, a poly-algorithmic number of operations at a time, over future insertions. Therefore, at any instant a number of such processes could be underway.

**6.4. Updating codes.** We consider the updates that need to be made to the various codes as a result of changes made to the intervals by *Assign*( ), *Reassign*( ), *Rescale*( ), and *Compact*( ) processes (as described in section 6.3).

First, consider an *Assign*$(\pi')$ process initiated by some node $x$, which is the head node on $i$-path $\pi$ and becomes the first node on $(i+1)$-path $\pi'$. *Assign*$(\pi')$ must assign a new interval to $\pi'$ and a new interval to $\pi$ nested within the interval for $\pi'$. It must then change the codes at all nodes in $T_x$ to reflect the change to the above two intervals. This is done as follows. The old separator string for $\pi$ in the codes at nodes in $T_x$ (including $x$ itself) will be updated to the new separator string for $\pi$. In addition, the separator string for $\pi'$ will be incorporated into the codes at all nodes in $T_x$. Thus the effect of *Assign*$(\pi')$ on the codes in $T_x$ is to make $\pi'$ appear as a path in their codes, but as a path with no primary nodes. *Reassign*$(x)$ will perform the changes needed to make $x$ a primary node on $\pi'$.

Next, consider a *Reassign*$(x)$ process. It is initiated when $x$ is in the process of leaving $i$-path $\pi$ on which it is currently primary (recall $x$ is also secondary on $(i+1)$-path $\pi'$ in this scenario). The process must remove the separator string for $\pi$ from the code at $x$ and from the codes at all the nodes in the subtree rooted at that child $y$ of $x$ which is not on $\pi$. In addition, this process must assign a new separator string for the path containing $y$ and modify the codes at all nodes in the subtree rooted at $y$ to reflect this.

We need to specify the scheduling of *Reassign*$(x)$ in more detail. When *size*$(x)$ reaches $2^{i+1}$, *Reassign*$(x)$ is made pending. At some future point, *Reassign*$(x_i)$ processes, $1 \le i \le k$, are all initiated, where nodes $x_1, x_2, \ldots, x_k$ are all the nodes currently primary on $\pi$ and secondary on $\pi'$. These *Reassign*( ) processes are performed in top-to-bottom order (i.e., if the nodes $x_1, x_2, \ldots, x_k$ are in top-to-bottom order, then *Reassign*$(x_1)$, *Reassign*$(x_2), \ldots,$ *Reassign*$(x_k)$ are performed in turn). This collection of *Reassign* processes is called a *Reassign* superprocess.

A *Compact*$(\pi)$ process for an $i$-path $\pi$ must assign a new separator code to all child paths of primary nodes on $\pi$ (secondary nodes are themselves contained in a child path of a primary node on $\pi$ by Lemma 6.2). *Compact*$(\pi)$ must also update codes at all nodes in the subtree rooted at the head of $\pi$ (other than codes for primary nodes on $\pi$). If $\pi$ happens to be a zero length path, the above description applies to its one or two child paths.

Finally, a *Rescale*$(\pi)$ process must assign a new interval $int_\pi$ to $\pi$ contained within $int_{\pi'}$, where $\pi'$ is the parent of $\pi$. In addition, the process must update the codes of all nodes in $T_{head(\pi)}$, replacing the old separator for $\pi$ with the new separator (corresponding to the change to interval $int_\pi$).

In addition to updating the codes, it is also necessary to update the annotations on the codes; recall the annotations label each bit in the code with the name of the centroid path whose separator contains this bit (also note the tagging mentioned in section 4.1). This can be done in $O(\log n)$ time per node.

We have now described the overall structure of the algorithm. For each unit weight increase in a node (from 0 to 1), which occurs as a node is inserted, $O(\log n)$ work is done in updating the ancestor centroid paths of the inserted node and initiating a constant number of *Assign*, *Rescale*, *Reassign*, and *Compact* processes for each such centroid path. A further $O(\log n)$ work is done to construct the code for the inserted node along with the annotations, using the code for its parent or child (this is explained in section 6.5). The work needed to be done on this insertion in order to further the progress of unfinished processes will be described in section 6.6. This part of the algorithm is what leads to the $O(\log^3 n)$ bound. Before describing this work, we introduce one more crucial aspect of the algorithm and also some crucial invariants which we will maintain.

*Two codes instead of one.* Even though all of the above description has been in terms of one code per node, we will actually maintain not one but two codes at each node. We will refer to these two codes as the *first code* and the *second code*. The reason for two codes is the following.

Consider two nodes $x, y$ and any process which needs to modify codes at both these nodes. At some intermediate time instant, this process could have modified the code at $x$ but not at $y$; as a consequence, the first bit of difference will no longer be related to the LCCP of $x$ and $y$. To make matters worse, there could be several such processes which have modified codes at one but not both of $x$ and $y$ (actually, our algorithm will ensure that there is only one such process).

To ensure that LCAs are indeed related to the very first bit of difference, we will maintain two codes instead of one at each vertex. Each process will be required to update both codes for all vertices of interest. However, all first codes will be updated first, and all second codes will be updated only after all first codes have been updated. The critical property is that at any instant of time, either both the first codes at $x, y$ would have been modified, or neither second code at $x, y$ has been modified. Thus either the first codes or the second codes will retain the relationship of the leftmost difference bit to the LCCP at each instant of time.

In what follows, unless explicitly stated, we will refer to both the codes for node $x$ collectively as the code at node $x$, or as *code*$(x)$.

**6.5. Invariants maintained.** To ensure that queries are answered correctly at every instant, we schedule the various processes so that the two invariants described below are maintained.

Invariant 1 states that each process finishes before the situation which caused the initiation of the process changes too dramatically. Some additional terminology will be helpful.

Definition 6.5. *A process associated with an i-path is called an i-process (i-Assign, etc.).*

Definition 6.6. *The size of path $\pi$, $size(\pi)$, is defined to be $|T_{head(\pi)}|$.*

Definition 6.7. *A weight increase is in the domain of path $\pi$, or into $\pi$ for short, if it is applied to a node in the subtree rooted at the current head of $\pi$.*

*Invariant* 1. Each process associated with $i$-path $\pi$ completes within a weight increase of $2^{i-3}$ into $\pi$ from the time the process was initiated. In addition to this, the following rules apply:

(a) *Assign*$(\pi)$ is initiated when $size(\pi)$ reaches $2^i$. (As $size(\pi)$ may never be exactly $2^i$, it is helpful to pretend that time is continous and that the weight increases occur continuously, and then we can define the initiation of *Assign*$(\pi)$ to occur exactly when $size(\pi) = 2^i$.)

(b) *Compact*$(\pi)$ is initiated following each weight increase of $2^{i+1}$ into $\pi$ from the time of $\pi$'s creation (i.e., *Assign*$(\pi)$'s initiation).

(c) Pending *Reassign*$(x)$ processes associated with $\pi$ are initiated following weight increase $h\,2^{i-2} + 2^{i-3}$ into $\pi$, from the time of $\pi$'s creation, for each integer $h \geq 4$.

(d) *Rescale*( ) is initiated when $size(\pi)$ reaches $2^{i+1} - 2^{i-3}$.

Corollary 6.8. *A Reassign$(x)$ process associated with $(i+1)$-path $\pi'$, which removes $x$ from i-path $\pi$, becomes pending when $size(\pi) = 2^{i+1}$ and $head(\pi) = x$, and completes before a further weight increase of $2^{i-1} + 2^{i-2}$ into $\pi$. In addition, if a Reassign$(z)$ process is created when node $z$ is inserted as the parent of node $x$ with an already pending Reassign$(x)$ process, the Reassign$(z)$ completes before the Reassign$(x)$ process completes.*

*Finally, $size(\pi) < 2^{i+1} + 2^{i-1} + 2^{i-2}$.*

*Proof.* *Reassign*$(x)$ becomes pending when $x$ becomes secondary on $\pi'$, i.e., when $size(\pi) = 2^{i+1}$ with $head(\pi) = x$. By Invariant 1(c) applied to $\pi'$, *Reassign*$(x)$ is initiated before a further weight increase of $2^{i-1}$ into $\pi'$ and completes within another weight increase of $2^{i-2}$ into $\pi'$. Any weight increase into $\pi$ during this time is also into $\pi'$. The first claim follows. The claim about *Reassign*$(z)$ follows due to the scheduling of *Reassign*( ) processes in top-to-bottom order.

To obtain the bound on $size(\pi)$ we show that $size(\pi)$ is less than $2^{i+1} + 2^{i-1}$ at the moment when a bunch of *Reassign*( ) processes removing nodes from $\pi$ are initiated. Let $x$ be the highest node on $\pi$ not being reassigned. Then, at that moment, $|T_x| < 2^{i+1}$. Following a weight increase of $2^{i-1}$ into $\pi'$, all the nodes being reassigned have been removed, and the next bunch of *Reassign*( ) processes is being initiated; at that moment again $size(\pi) < 2^{i+1} + 2^{i-1}$. The maximum size for $\pi$ therefore occurs just before the *Reassign*( ) processes are completed, i.e., just before a weight increase of $2^{i-2}$ into $\pi'$ (and hence into $\pi$) from the moment the *Reassign*( ) processes are initiated. This yields the claimed bound on $size(\pi)$. ☐

Corollary 6.9. *For each path $\pi$, there is at most one Assign$(\pi)$, Rescale( ), Reassign$(x)$ for $x$ secondary on $\pi$, or Compact$(\pi)$ under way at any given time.*

*Invariant* 2. For any pair of nodes $x, y$ there is at most one *Assign*( ), *Reassign*( ), *Rescale*( ), or *Compact*( ) process which must modify the codes at both $x$ and $y$ and which has modified one but not both the first codes at these nodes. Similarly, there is at most one process which must modify the codes at both $x$ and $y$ and which has modified one but not both the second codes at these nodes. Finally, if a process with the former description exists, then a process with the latter description cannot exist.

We remark that Invariant 1 is not hard to maintain. Similarly, Invariant 2 is easy to maintain using a simple blocking mechanism for processes. However, maintaining both invariants simultaneously is nontrivial. This is because weight increases in different parts of the tree may occur at different rates and unfinished processes will therefore be driven at different speeds by these weight increases. In particular, a process could be blocked indefinitely. One solution to this problem is to make a blocked process help the blocking process. This will be described in detail in section 6.6.

We are now ready to show that the intervals are sufficiently large.

LEMMA 6.10. *Suppose Invariant* 1 *holds and that* $c' \geq 5$ *and* $c = 2c'$. *Let* $\pi$ *be an* $i$*-path.*

(a) *The insertion side of* $int_\pi$ *cannot be filled up prior to the completion of* $Rescale(\pi)$ *(recall that* $int_\pi$ *has size* $2^{ic}$ *before* $Rescale(\pi)$ *completes).*
(b) *The insertion side of* $int_\pi$ *following the completion of* $Rescale(\pi)$ *will not fill up before the first initiation of* $Compact(\pi)$ *(*$int_\pi$ *now has size* $2^{ic+c'}$*).*
(c) *Consider an initiation of* $Compact(\pi)$. *Over a subsequent weight increase of* $2^{i+1}$ *into* $\pi$, *the side of* $int_\pi$ *being filled by* $Compact(\pi)$ *will have room for the subintervals being added to* $int_\pi$.

*Proof.* The proof uses an induction on $i$. The result is trivially true for $i = 0$. Thus the inductive step remains to be proven. We start with part (a). Up to the completion of $Rescale(\pi)$, $size(\pi)$ remains less than $2^{i+1}$. We show that all the subintervals that could be generated until $Rescale(\pi)$ completes will fit into $int_\pi$.

We sum the length of all the subintervals inserted into $int_\pi$ since the initiation of $Assign(\pi)$. Some of these intervals may have been deleted by the time $Rescale(\pi)$ completes. The length of each interval that is inserted is at most $2^{(i-1)c+c'}$. A particular subtree rooted at an off-path node could repeatedly delete and insert subintervals increasing in size by successive factors of $2^{2c'}$. Thus such a subtree, whose root is a primary node in an $h$-path, could be responsible for a total subinterval of length $\sum_{j=0}^{2h+1} 2^{jc'} < \frac{2^{(2h+2)c'}}{2^{c'}-1}$. The total length of gaps between subintervals is at most the total length of the subintervals. Since $Rescale(\pi)$ completes following a weight increase of $2^i$ into $\pi$ from the time $\pi$ was created, the total length of $int_\pi$ occupied by deleted and undeleted subintervals and the gaps between them is at most $2\sum_r \frac{2^{c'(2h_r+2)}}{2^{c'}-1}$, where $\sum_r 2^{h_r} \leq 2^{i+1} = 4 \cdot 2^{i-1}$ and $h_r \leq i-1$. This is at most $\frac{8 \cdot 2^{ic'}}{2^{c'}-1} \leq \frac{2^{2ic'}}{2}$ for $c' \geq 5$.

The proof of part (c) is broadly similar. W.L.O.G., we assume that $Compact(\pi)$ inserts into the right half of $int_\pi$ (now $int_\pi$ has size $2^{ic+c'}$). We show that from the time of the initiation of $Compact(\pi)$, the subintervals in $int_\pi$ when $Compact(\pi)$ was initiated together with any new subintervals inserted up until the next initiation of $Compact(\pi)$ will all fit in the right half of $int_\pi$. When $Compact(\pi)$ is initiated, $size(\pi) < 2^{i+1} + 2^{i-1} + 2^{i-2}$; the next initiation of $Compact(\pi)$ occurs following a weight increase of $2^{i+1}$ into $\pi$. The length of each subinterval inserted into $int_\pi$ is at most $2^{2ic'}$; such a subinterval would be due to a subtree of size at least $2^i$. Similarly to before, the total length of subintervals for which such a subtree could be responsible is less than $2^{2ic'} + \sum_{j=0}^{2i-1} 2^{jc'} < \frac{2^{(2i+1)\,c'}}{2^{c'}-1}$. A smaller subtree, whose root is a primary node on an $h$-path could contribute subintervals of total length no more than $\frac{2^{(2h+2)\,c'}}{2^{c'}-1}$. As before, the total length of the right half of $int_\pi$ occupied by deleted and undeleted subintervals and the gaps between them is less than $\frac{2a \cdot 2^{(2i+1)\,c'}}{2^{c'}-1} + 2\sum_r \frac{2^{(2h_r+2)\,c'}}{2^{c'}-1}$, where $a \cdot 2^i + \sum_r 2^{h_r} < 2^{i+1} + 2^{i-1} + 2^{i-2} + 2^{i+1}$ and $h_r \leq i-1$. This is at most $(8 \cdot 2^{(2i+1)\,c'} + 4 \cdot 2^{2ic'})/(2^{c'}-1) \leq 12 \cdot 2^{(2i+1)\,c'}/(2^{c'}-1) \leq 2^{(2i+1)\,c'}/2$, if $c' \geq 5$.

The proof of part (b) is identical to that of part (c) except that we need only consider the intervals due to an initial weight of $2^i$ (when $\pi$ was created) and a weight increase of $2^{i+1}$ (namely, up to the time when *Compact*($\pi$) is first initiated). ◻

**6.6. Some details of processes.** Before proceeding, some details of *Assign*( ), *Reassign*( ), *Rescale*( ), and *Compact*( ) processes need to be carefully examined. These details arise because a process is performed over a sequence of future insertions. A fundamental issue here is that nodes can lie on two paths: on one as a primary node and on the other as a secondary node.

Our goal, which enables the $O(1)$ query procedure, is to establish the following claim.

CLAIM 6.11. *The changes due to $x$ becoming primary on path $\pi'$ and ceasing to be primary on path $\pi$ are reflected in the code at a descendant node $y$ of $x$ only due to the modifications performed by the process carrying out $x$'s change of primary paths.*

*Process blocking.* To maintain Invariant 2, the following strategy is used. A process first updates all the first codes for the nodes it needs to process in preorder and then updates all the second codes, again in preorder. When the process begins work on the first codes of a subtree rooted at a node $z$ it will mark $z$ as blocked and will only remove the mark when it has finished updating the first codes in $z$'s subtree. It will follow the same blocking procedure when updating second codes. If a process $P$ comes to a blocked node it will not proceed with its work until the node is unblocked (later, we explain the helping of the blocking process undertaken by $P$ while it is waiting).

In addition, a process keeps the topmost node it is updating blocked until it is completed. Finally, a *Reassign*($x$) process, in addition to keeping $x$ blocked throughout its processing, will as its last step make $x$ primary on the path on which $x$ had been secondary.

We specify later just how a newly inserted node is marked.

*Constructing codes for newly inserted nodes.* Recall that each node has two associated codes, a first code and a second code. For a newly inserted node $x$, each code is obtained from the corresponding code for its parent or child as follows.

1. If $x$ is inserted as a child of a leaf node, then it is given the same code as its parent. This reflects the fact that $x$ lies on the same 0-path as its parent. Of course, if and when $x$'s weight increases sufficiently, its parent will leave this 0-path.
2. If $x$ is inserted as a leaf child of an internal node $v$, then it forms a new singleton path. A new interval is assigned for this path. The code for $x$ is just the code for its parent appended with the separator string for this new singleton path. There is one more complicated scenario, which arises when a *Reassign*($v$) process is underway, and the first code for $v$ has been updated, but the second has not. Let $\pi$ be $v$'s primary path and $\pi'$ its secondary path. In this case, $x$ is assigned two subintervals, one in $int_{\pi'}$ for its first code and one in $int_{\pi}$ for its second code. When the *Reassign*($v$) process updates second codes it will replace the subinterval in $x$'s second code with the subinterval in $int_{\pi'}$ used in its first code.
3. If $x$ is inserted as an internal node, then the code for $x$ is made identical to that for its child. This makes $x$ primary on the same path $\pi$ as its child. In addition, if $x$'s child is marked, then so is $x$. Of course, it may be that $x$ is also added to the parent path of $\pi$, in which case a *Reassign*($x$) process is created. There is one exception. Let $y$ be $x$'s child. If a *Reassign*($y$) is already underway, the code for $x$ is set to the updated code for $y$, i.e., the

code with separator($\pi$) removed, and in this case $x$ is not marked.

We mention here that Invariant 2 also applies to newly inserted nodes $x$ and $y$ which inherit their codes from partially processed nodes (i.e., nodes for which one but not both codes have been updated by some process).

*Process roots.* Each process must modify codes at certain nodes in the tree. The node closest to the root of $T_1$ whose code a process must modify is called the *root* of the process. Since nodes are inserted dynamically, the root of a process needs to be defined more precisely. For an $Assign(\pi)$, $Compact(\pi)$, or $Rescale(\pi)$ process, $head(\pi)$ is the root of the process; if a new node $z$ is inserted and becomes the head of $\pi$ as one of these processes is underway, $z$ becomes the new root of the process. While one of these processes is underway no node $x$ leaves $\pi$ due to a $Reassign(x)$ process until the $Assign(\pi)$, $Rescale(\pi)$, or $Compact(\pi)$ has completed; this is a consequence of the blocking strategy. For a $Reassign(x)$ process, the root is always $x$.

*Helping a blocking process.* If a process $P$ reaches a marked node $x$, it will seek to help an unblocked process in $T_x$ so as to advance the processing that will lead to the removal of the mark on $x$. To this end it traverses a path of marked nodes from $x$; at the last marked node $y$, it discovers the process $Q$ that marked $y$ and performs the next code update for $Q$. This may entail unmarking $O(\log n)$ nodes and marking up to one node (since $Q$ marks a nonleaf node prior to updating it). To facilitate this process, each mark includes the name of the process making the mark. As it suffices to have $P$ help some unblocked process on which it is waiting either directly or indirectly, it suffices that $P$ traverse a maximal path of marked nodes in $T_x$; thus this process takes $O(\log n)$ time. It is called a *basic step.*

$i$-process $P$ could be blocked by an ancestral process or by a descendant process. We will need to ensure that $P$ is blocked by at most one ancestral process. Further, this only happens at $P$'s initiation. If $P$ is so blocked, it puts a subsidiary mark on its root. The meaning of this mark is that as soon as $Q$'s mark is removed, where $Q$ is the blocking process, $P$'s mark is then instantiated. As there is only one active process per path, there are at most two such subsidiary marks per node (two paths can share a root, either because a new path has only secondary nodes and hence shares its root with a child path, or because a path temporarily has no nodes; the nodeless path will use its parent node as its root for any associated processes). In the case that two marks are present, the mark for the deeper path will be instantiated. It remains the case that each process $P$ is blocked by at most one ancestral process.

*Process interleaving and its effect on code updates.* We need to examine the interleaving of processes for paths $\pi$ and $\pi'$, $\pi'$ a child of $\pi$, and how this may affect the update of code portions corresponding to $sep(\pi)$ and $sep(\pi')$.

Because of their relative timing, $Compact(\pi)$, $Rescale(\pi)$, $Assign(\pi)$, and $Reassign(x)$ for $x$ secondary on $\pi$ do not overlap.

$Compact(\pi)$ and $Reassign(x)$ for $x$ secondary on $\pi$, update only the portion of the code corresponding to $sep(\pi')$ for $\pi'$ a child of $\pi$; $Rescale(\pi)$ updates only the portion of the code corresponding to $sep(\pi)$; $Assign(\pi)$, on the other hand, updates the portions of the code corresponding to both $sep(\pi)$ and $sep(\pi')$, where $\pi'$ is the primary path for nodes secondary on $\pi$. Thus we will need to consider the possible interaction of $Compact(\pi)$ and $Assign(\pi')$ or $Rescale(\pi')$, and of $Reassign(x)$ and $Assign(\pi''')$ or $Rescale(\pi''')$ where $x$ is secondary on $\pi$, primary on $\pi'$, and $\pi'''$ is the off-path child of $x$ (w.r.t. $\pi'$).

Consider such a $Compact(\pi)$ process. Let $y$ be an off-path child of $\pi$, secondary on $\pi'$ and primary on $\pi''$. So there is an $Assign(\pi')$ process at hand. If the $Compact(\pi)$ process updates $code(y)$ first, then $\pi''$ receives a new subinterval in the insertion half of $int_\pi$, the right half say; subsequently, the $Assign(\pi')$ gives $\pi'$ a new subinterval in the right half of $int_\pi$. On the other hand, if the $Assign(\pi')$ process updates $code(y)$ first, but after the $Compact(\pi)$ process has been initiated, then the $Assign(\pi')$ process gives $\pi'$ a new subinterval in the right half of $int_\pi$, and subsequently the $Compact(\pi)$ process does nothing further to $int_{\pi'}$ and the codes for nodes in $T_y$. The possible interactions of $Compact(\pi)$ and $Rescale(\pi')$ are identical.

Next, we consider a $Reassign(x)$ process for a node $x$, primary on $\pi'$ and secondary on $\pi$. Let $y$ be the child of $x$ which is not on $\pi'$. Let $\pi''$ be the path containing $y$. After the creation of the $Reassign(x)$ process suppose that $y$ becomes secondary on a new centroid path $\pi'''$. $\pi'''$ now becomes the parent of $\pi''$ and an $Assign(\pi''')$ process is initiated. We consider two cases in turn.

First, suppose that $Assign(\pi''')$ modifies the code for $y$ before $Reassign(x)$ does so. Then the separator for $\pi'$ will be in $code(y)$ when it is processed by $Assign(\pi''')$. Thus $int_{\pi'''}$ will be assigned so that it is nested within $int_{\pi'}$. At some subsequent point, $Reassign(x)$ will remove the separator string for $\pi'$ from $code(y)$ and reassign $int_{\pi'''}$ so that it is nested within $int_\pi$.

Second, suppose that $Reassign(x)$ has updated $code(y)$ before $Assign(\pi''')$. Then, when $Assign(\pi''')$ processes $code(y)$ it no longer contains the separator for $\pi'$. The behavior of $Assign(\pi''')$ is now as expected with $int_{\pi'''}$ being assigned so as to be nested within $int_\pi$. But note that when $Reassign(x)$ processed $code(y)$, $code(y)$ indicated that $y$ was primary on $\pi''$. So $\pi''$ was reassigned a new subinterval within $int_\pi$ and the resulting separator string was included within $code(y)$. At some later instant, $Assign(\pi''')$ included the separator string for $\pi'''$ in $code(y)$ and replaced the current separator string for $\pi''$ with a new separator string corresponding to a subinterval nested within $int_{\pi'''}$. Ultimately, the $Reassign(y)$ process created by $y$ becoming secondary on $\pi'''$ removed the latter separator string from $code(y)$. Again, the possible interactions of $Reassign(x)$ and $Rescale(\pi''')$ are identical.

Consider Claim 6.11. Note that it would not hold if, in the first case above, $Assign(\pi''')$ assigned an interval to $\pi'''$ nested within the interval for $\pi$ and incorporated the associated separator into $code(y)$. For if $Assign(\pi''')$ did so then the change resulting from $x$ becoming primary on $\pi$ would be reflected in $code(y)$ by a modification made by $Assign(\pi''')$ and not by $Reassign(x)$.

It is possible for the updates described above involving two processes ($Compact(\pi)$ and $Assign(\pi')$ or $Rescale(\pi')$, $Reassign(x)$ and $Assign(\pi''')$ or $Rescale(\pi''')$) to occur in a different order on the two codes. The only possible interleaving for the $Compact(\pi)$ process is that first $Compact(\pi)$ updates the first codes of nodes in $T_y$, then $Assign(\pi')$ (or $Rescale(\pi')$) updates both codes of nodes in $T_y$, and then $Compact(\pi)$ examines $T_y$ but makes no further updates to the codes. For the $Reassign(x)$ process, the only possible interleaving is that first $Reassign(x)$ updates the first codes of nodes in $T_y$, then $Assign(\pi''')$ (or $Rescale(\pi''')$) updates both codes of nodes in $T_y$, the $Reassign(x)$ updates the second codes of nodes in $T_y$. Each update follows the appropriate rules for the codes it encounters which will differ for the first and second codes. All we have to ensure is that the subintervals chosen for the second codes within a particular interval are the same as those selected for the first codes within the same interval. Thus, for example, in the $Reassign(x)/Assign(\pi''')$ scenario above, $Assign(\pi''')$ assigns a separator for $\pi'''$ to $code(y)$, $(sep(\pi'''))^1$ say, contained in $int_\pi$, while for the second code it assigns a separator $(sep(\pi'''))^2$ for $\pi'''$ contained within $int_{\pi'}$, and then $Reassign(x)$ replaces this separator with $(sep(\pi'''))^1$.

**6.7. Processing queries.** The algorithm ensures that the first bit of difference between either the first codes at $x$ and $y$ or the second codes at $x$ and $y$ is related to the LCCP, which itself can be obtained using the algorithm detailed below. The first step is to find the leftmost bit of difference $d_{first}$, between the first codes at $x$ and $y$, and thereby to find the rightmost path $\pi_{first}$ such that the first codes at $x$ and $y$ agree on all separators up to and including that for $\pi_{first}$. Similarly, $d_{second}$ and $\pi_{second}$ are identified with respect to the second codes at $x$ and $y$.

Consider $z_{first} = head(\pi_{first})$ and $z_{second} = head(\pi_{second})$. As we will see, if the first code of $z_{first}$ agrees with the first code of $x$ (and of $y$) on all separators up to $\pi_{first}$ and the same is true for $z_{second}$, then one of $\pi_{first}$ and $\pi_{second}$ is the LCCP of $x$ and $y$; if it holds only for $z_{first}$, then $\pi_{first}$ is the LCCP, and similarly for $z_{second}$.

LEMMA 6.12. *If a process $P$ updates the separator $sep(\pi)$ for a path $\pi$, then $sep(\pi)$ is identical either for all first codes for nodes in $T_{head(\pi)}$ or for all second codes for nodes in $T_{head(\pi)}$, and when $P$ unmarks $T_{head(\pi)}$ it is identical both for all first codes for nodes in $T_{head(\pi)}$ and for all second codes, although its encoding in the first and second codes may differ.*

*Proof.* The proof uses an induction on time. So assume that the result is true at the moment $P$ marks $T_{head(\pi)}$. Any process that updates $sep(\pi)$ must mark $T_{head(\pi)}$, thus until $P$ unmarks $T_{head(\pi)}$, only $P$ can update $sep(\pi)$ in the codes for nodes in $T_{head(\pi)}$. The lemma follows. □

*Comment.* The interleaving described in the previous section shows that the two codes may differ following completion of a process, though when the second process touching these codes also completes, the equality will be restored.

LEMMA 6.13. *If $sep(\pi)$ appears in $code(x)$, then $head(\pi)$ is an ancestor of $x$.*

*Proof.* The proof is by induction on time. When a node is inserted, if a leaf it inherits its code from its parent and thus the claim is true at this point; if an internal node, it inherits its code from its child and the claim is true at this point too.

An update which changes $head(\pi)$ will occur only after $sep(\pi)$ is appropriately updated in the codes for all descendants of $head(\pi)$ and thus the claim continues to hold. □

Let $z$ be the LCA of $x$ and $y$, and suppose that $z$ is primary on $\pi$ and let $w = head(\pi)$. Then we have the following.

LEMMA 6.14. *Suppose either that $w$ is not marked, or if $w$ is marked, then W.L.O.G. it is the second codes in $T_w$ that are begin updated. Then $z$ and $z_{first}$ lie on the same path. Also, the first code of $z_{first}$ agrees with the first code of $x$ on all the separators up to $\pi_{first}$. Finally, $\pi_{first}$ is the LCCP of $x$ and $y$.*

*Proof.* By Lemma 6.12, any encoding $sep(\widetilde{\pi})$ for path $\widetilde{\pi}$ appearing in the first code of $z$ must also appear in the first codes of $x$ and $y$. Thus $z_{first}$ lies on the same path as $z$ or is a descendant of $z$. But clearly if $sep(\widetilde{\pi})$ appears in $code(x)$, then there is a node $v$ on $\widetilde{\pi}$ with $v$ an ancestor of $x$ ($v$ may be primary or secondary on $\widetilde{\pi}$). Thus there are nodes on $\pi_{first}$ ancestral to each of $x$ and $y$, which must include $z_{first} = head(\pi_{first})$. Thus $z_{first}$ and $z$ lie on the same path. □

COROLLARY 6.15. *One of $\pi_{first}$ and $\pi_{second}$ is the LCCP of $x$ and $y$.*

By Lemma 6.13 $\pi_{first}$ and $\pi_{second}$ are both ancestral to each of $x$ and $y$. Thus if $\pi_{first} = \pi_{second}$ they are both the LCCP. Otherwise, let $\pi_{first}$ be a $j$-path and $\pi_{second}$ a $k$-path. The one with the larger index (among $j$ and $k$) provides the LCCP.

This yields the $O(1)$ algorithm for finding the LCCP and hence the LCA.

**6.8. Running time.** In order to maintain Invariant 2 while meeting Invariant 1, a process may need to help (perform the work of) other processes that are blocking it.

Thus the main issue is to determine how much work a process may do. We measure this work in *basic steps*.

DEFINITION 6.16. *A basic step of a process is the traversal of $O(\log n)$ edges of the subtree it is processing followed by the updating of one of the codes at a single node in this subtree. (The final basic step entails only edge traversals.)*

CLAIM 6.17.

(i) *A basic step takes $O(\log n)$ time to perform.*

(ii) *A process rooted at node $x$ performs at most $2|T_x| + 1$ basic steps on its task, where $|T_x|$ is the size of tree $T_x$ in vertices immediately before the completion of the process.*

LEMMA 6.18. *An Assign$(\pi)$, Compact$(\pi)$, Rescale$(\pi)$, or Reassign superprocess associated with class $i$ path $\pi$ performs at most $e \cdot 2^i \cdot i$ basic steps on itself and all the tasks it helps, for a suitable constant $e > 0$.*

Lemma 6.18 depends on a scheduling algorithm which ensures that for each weight increase of $2^{i-3}$ into path $\pi$, at least $e \cdot i \cdot 2^i$ basic steps are performed on process $P$ and the tasks it helps, where $P$ is associated with path $\pi$. We describe the scheduler later. Clearly, if the scheduler exists, then the truth of Lemma 6.18 up to a given time immediately implies Invariant 1 also holds up to that time.

*Proof of Lemma* 6.18. The proof uses a double induction, the outer induction being on $t$, the weight increase since the data structure was initiated (time for short), and the inner induction being on the class $i$ of the $i$-path $\pi$. Note that time proceeds in increments of $1/\log^3 n$. Our proof depends on the following claim, whose proof uses the inductive hypothesis. □

CLAIM 6.19. *Suppose Lemma* 6.18 *holds through time $t$ and for processes associated with $h$-paths, $h < i$, at time $t + 1/\log^3 n$. Then consider an $i$-path $\pi$ at time $t + 1/\log^3 n$ with head $x$. Consider the collection of all $h$-processes, $h < i$, having roots in $T_x$ which have been activated by time $t + 1/\log^3 n$. The total number of basic steps that have been performed on all these processes through time $t + 1/\log^3 n$ since their first activation is $O(i \cdot 2^i)$.*

*Proof of Claim* 6.19. We first note that if Lemma 6.18 holds for a given process $P$ associated with an $i$-path at its termination, then the size of the subtree rooted at $P$'s root at its termination is less than $2^{i+1} + 2^{i-1} + 2^{i-2}$, by Corollary 6.8. Consequently, if a process associated with an $i$-path is not complete at a time $t'$ for which Lemma 6.18 holds, then the subtree rooted at the process root has size less than $2^{i+1} + 2^{i-1} + 2^{i-2}$. In particular, the subtree rooted at $P$'s root has size less than $2^{i+1} + 2^{i-1} + 2^{i-2}$ at time $t$, and hence size less than $2^{i+1} + 2^{i-1} + 2^{i-2} + 1/\log^2 n$ at time $t + 1/\log^3 n$.

Now, we bound the number of basic steps performed by each type of process having its root in $T_x$, where $x$ is $P$'s root.

*Assign( ) processes.* There is one *Assign*( ) process for each path inside $T_x$. By the inductive hypothesis, for an *Assign*$(\pi')$ process associated with $h$-path $\pi'$, $h < i$, the associated subtree has size less than $2^{h+1} + 2^{h-1} + 2^{h-2}$ at time $t + 1/\log^3 n$ and hence the *Assign*$(\pi')$ process has had at most $O(2^h)$ basic steps performed on it. For each $h < i$, each $h$-path has a distinct set of nodes of combined size at least $2^h$ associated with it, namely either the subtree rooted at the head of the path, or if the $h$-path $\pi'$ has $h$-path $\pi''$ as a child, the associated nodes are given by $T_{head(\pi')} - T_{head(\pi'')}$. Thus there are at most $(2^{i+1} + 2^{i-1} + 2^{i-2} + 1/\log^2 n)/2^h$ $h$-paths in $T_x$, and the total number of basic steps performed through time $t + 1/\log^3 n$ on their *Assign*( ) processes is $O(2^i)$. Summing over all $h < i$ gives a bound of $O(i \cdot 2^i)$ on the number of basic steps performed on *Assign*( ) processes for $h$-paths, $h < i$, inside $T_x$.

*Rescale*( ) *processes.* The analysis is identical to that for the *Assign*( ) processes.

*Compact*( ) *processes.* Recall that successive *Compact*($\pi'$) processes for $h$-path $\pi'$ are separated by weight increases of $2^{h+1}$ into $\pi'$. Likewise the first *Compact*($\pi'$) process occurs following a weight increase $2^{h+1}$ from the initiation of *Assign*($\pi'$). Further, each weight increase at a node contributes only to the weight increase for paths that are the node's ancestors, hence for at most two $h$-paths for each value of $h$. Thus at most $2(2^{i+1}+2^{i-1}+2^{i-2}+1/\log^2 n)\,/2^{h+1}$ *Compact*($\pi'$) processes have been activated for $h$-paths $\pi'$ contained in $T_x$. By the inductive hypothesis, these processes have each had at most $O(2^h)$ basic steps performed on them by time $t+1/\log^3 n$, and hence summing over all $h < i$, and all paths in $T_x$, yields a bound of $O(i \cdot 2^i)$ basic steps performed on the *Compact*( ) processes for $h$-paths, $h < i$, inside $T_x$.

*Reassign*( ) *processes.* The argument is very similar to that for the *Compact*( ) processes, with each bunched *Reassign*( ) superprocess resembling a *Compact*( ) process in its cost. We account for a bunched $h$-superprocess by associating it with the at least $2^{h-2}$ insertions to its associated $h$-path $\pi'$ from either the time of the creation of $\pi'$ or the start of the previous *Reassign*( ) superprocess associated with $\pi'$, whichever is more recent, to the moment the current superprocess begins to be processed.

Consider a weight increase; it is charged for the *Reassign*( ) superprocesses at its ancestors, i.e., for at most two *Reassign*( ) $h$-superprocesses, for each $h$. It follows that there have been at most $2(2^{i+1}+2^{i-1}+2^{i-2}+1/\log^2 n)\,/2^{h-2}$ such superprocesses activated in $T_x$. Summing over all paths and $h < i$, we conclude that a total of $O(i\cdot 2^i)$ basic operations have been performed on the *Reassign*( ) superprocesses with roots in $T_x$.

This concludes the proof of Claim 6.19.

We now complete the proof of Lemma 6.18.

Consider the currently active process $P$ associated with $i$-path $\pi$, if any. By Claim 6.19, $P$ has performed at most $O(i \cdot 2^i)$ basic operations helping processes associated with $h$-paths, $h < i$. Since $P$ has not completed at time $t$, as already noted, $|T_x| < 2^{i+1} + 2^{i-1} + 2^{i-2} + 1/\log^2 n$ at time $t + 1/\log^3 n$. Thus the number of basic steps performed on $P$'s task and the up to one other task it may help associated with some ancestral $j$-path, $j > i$, is $O(2^i)$. This gives a total bound of $O(i \cdot 2^i)$ on the basic steps performed by $P$. □

We must still describe the processing performed following a weight increase. Following a $\Theta(1/\log n)$ weight increase at a node, the size of each of its $O(\log n)$ ancestral central paths are incremented, up to $O(\log n)$ new secondary nodes are added to the paths to which they newly belong, and up to $O(\log n)$ new processes are initiated. Then $e$ basic steps are performed on the active process at each ancestral centroid path, if any, for a total of $O(\log n)$ basic steps, which takes $O(\log^2 n)$ time. Node insertion costs $O(\log n)$ per node, but there are $O(1)$ node insertions per unit weight increase, so this is relatively insignificant.

The $O(1)$ time algorithm requires a more elaborate scheduling procedure for two reasons. First, we cannot keep the recorded sizes of the centroid paths completely up to date and so processes may be late in getting underway, and second, we do not want to have multiple basic steps partially completed. This leads us to perform basic steps over a $\Theta(1/\log^2 n)$ weight increase once started, even if the weight increase is not all occuring in the relevant subtree. Our scheduling procedure is based on a variant of the Dietz–Sleator "cup-filling" methodology [DS87], which we develop in the next section.

**6.9. Dietz–Sleator "cup-filling" with dated priorities.** We seek to handle a task scheduling scenario of the following flavor. There are at most $k$ tasks at any one time. Tasks have associated priorities which increase, possibly at different rates, as they are delayed. Furthermore, the known priorities may be somewhat dated and hence potentially inaccurate. Our goal is to determine conditions under which the following strategy is effective: schedule for *atom* steps the task with current highest known priority and iterate.

So let $\Gamma = \{P_1, P_2, \ldots\}$ be a collection of no more than $k$ tasks. Suppose each task is performed in atomic chunks of length *atom* and suppose each task has length at most $\ell$, an integer multiple of atom. Task $P_i$ has an associated priority $p_i \geq 0$. Priorities only increase. At any time a new task of priority 0 may be created so long as there are at most $k$ tasks altogether. It will be convenient to keep placeholder tasks of priority 0 to ensure that there are always exactly $k$ tasks at hand.

After every atom steps a task $P_i$ is chosen to be executed for the next *atom* steps (possibly, but not necessarily, the same task as on the previous *atom* steps) which satisfies the following rule:

$$\lambda p_i - \text{work performed on } P_i + \lambda \cdot \mathit{error} \geq \lambda p_j - \text{work performed on } P_j \text{ for all } j \neq i,$$

where *error* represents the maximum error in the recorded priorities (as opposed to the actual priorities $p_i$, $p_j$) and $\lambda > 0$ is a scale parameter relating priorities to steps executed. After executing these *atom* steps, the priorities of an arbitrary subset of tasks are increased by a combined total of at most *p-inc*.

LEMMA 6.20. *The above scheduling algorithm, if atom* $\geq 4\lambda \max\{\mathit{error}, \textit{p-inc}\}$, *satisfies*

$$\lambda p_i + \ell - \textit{work performed on } P_i \leq \lambda\,(\mathit{error} + \textit{p-inc}) + (\mathit{atom} + \ell) \\ + 4\lambda(\mathit{error} + \textit{p-inc}) \log k.$$

COROLLARY 6.21. $p_i \leq (\mathit{error} + \textit{p-inc}) + 1/\lambda(\mathit{atom} + \ell) + 4(\mathit{error} + \textit{p-inc}) \log k.$

*Proof of Lemma* 6.20. We use a potential argument. We show that if there is a task $P_j$ with priority plus $1/\lambda$ times remaining work (strictly, $1/\lambda(\ell -$ work performed on $P_j))$ at least $\ell/\lambda + \mathit{error} + \mathit{atom}/\lambda$, when a task is chosen to have *atom* steps executed, then the potential will not increase following these *atom* steps being performed and the applying of the combined *p-inc* increment to the priorities.

We associate potential $c^{r_i}$ with task $P_i$, $1 \leq i \leq k$, where $c > 1$ is a suitable constant and $r_i$ is defined as follows: $r_i = \lambda p_i + \ell -$ work performed on $P_i$.

Clearly, following *atom* steps being executed on $P_i$ and potentials being incremented by *p-inc*, the maximum increase in potential occurs if all the incremental priority is concentrated on the task $P_j$ with largest $r_j$. We note that $r_i + \lambda \cdot \mathit{error} \geq r_j$ prior to the execution of *atom* steps on $P_i$. Thus if $r_j \geq \ell + \lambda \cdot \mathit{error} + \mathit{atom}$, then $r_i \geq \ell + \mathit{atom}$, so after *atom* steps of work are performed on $P_i$, $P_i$'s potential decreases by a multiplicative factor of $c^{\mathit{atom}}$. We want to ensure that the potential does not increase. For this, it suffices that

$$c^{r_j + \lambda\, \textit{p-inc}} + c^{r_i - \mathit{atom}} \leq c^{r_j} + c^{r_i}.$$

Clearly, this is hardest to satisfy with $r_i + \lambda \cdot \mathit{error} = r_j$; so it suffices that

$$c^{\lambda(\mathit{error}\ +\ \textit{p-inc})} + c^{-\mathit{atom}} \leq c^{\lambda\ \cdot\ \mathit{error}} + 1.$$

Let $\Delta = \lambda \max\{error, \textit{p-inc}\}$. Choosing $c$ so that $c^{2\Delta} = \sqrt{2}$ and choosing $atom \geq 4\Delta$ yields a sufficient condition of

$$\sqrt{2} + (1/\sqrt{2})^2 \leq 1 + 1,$$

which is true.

Thus the largest potential possible is less than

$$(k-1)\, c^{\lambda error + (atom+\ell)} + c^{\lambda(error + \textit{p-inc}) + (atom+\ell)} \leq k\, c^{\lambda(error+\textit{p-inc})+(atom+\ell)}.$$

Hence $c^{r_i} \leq k\, c^{\lambda(error+\textit{p-inc})+(atom+\ell)}$ for all $i$, from which the claimed bound on $r_i$ follows. □

A special case arises when $error = 0$ and $atom = \ell$.

COROLLARY 6.22. *If $error = 0$ and $atom = \ell$, then $p_i \leq (9 + 4\log k)\,\textit{p-inc}$.*

*Proof.* Set $\lambda = atom/(4\textit{p-inc})$. □

Dietz and Sleator proved this bound with somewhat tighter constants, namely $p_i \leq \textit{p-inc} \cdot (\log k + O(1))$. This is often called the Dietz–Sleator cup-filling lemma.

**6.10. Scheduling in the $O(1)$ time algorithm.** In the $O(1)$ time algorithm a layered structure will be used. The tree is binarized and partitioned into subtrees of size $O(\log^3 n)$. The roots of these subtrees and their LCAs form an implicit tree on which the previous $O(\log^3 n)$ time update algorithm is run. Intuitively, the subtree roots change only every $\Theta(\log^3 n)$ insertions, which provides enough time to perform the $O(\log^3 n)$ operations needed for an update.

Let $T$ denote the tree of $n$ nodes on which LCA queries are being performed and let $T_1$ denote the implicit tree. As we will see, $T_1$ has at most $4n/\log^3 n$ nodes, and this relationship applies to each subtree of $T$ and the corresponding subtrees of $T_1$. An insertion of a node $v$ in $T$ will result in a weight increase of either 0 or $4/\log^3 n$ to the following node in $T_1$: the node that is the root of the size $O(\log^3 n)$ subtree containing $v$ in the binarized version of $T$. The rule for the weight increase is discussed later in section 7 when we discuss how to maintain the size $O(\log^3 n)$ subtrees.

Also, a new node of weight zero may be inserted in $T_1$ as a result of an insertion into $T$. As already noted, weight 0 nodes are adjacent only to weight 1 nodes. Again, details on when this happens are given in section 7.

At this point, we describe a schedule that updates path weights and performs *Assign*( ), *Rescale*( ), *Reassign*( ), and *Compact*( ) processes as needed but with only $O(1)$ work per insertion to $T$.

The major difficulty we face is that on updating the weight of a node (as a result of an insertion in $T$), we cannot immediately update the weights of all the ancestral centroid paths (note that we only track the weight of the subtrees rooted at the heads of centroid paths—together with the individual node weights, this suffices to track the weight changes when a head node leaves a centroid path). Instead we create a weight update task for each node in $T_1$. The weight update task for node $v$ is responsible for updating the weights of head nodes ancestral to $v$ to reflect an increase in $v$'s weight. It may be run multiple times. The execution of weight update tasks is alternated with the execution of *Assign*( ), *Rescale*( ), *Reassign*( ), and *Compact*( ) processes in stages of length $\Theta(\log n)$, with each update weight task being run to completion once initiated. An update weight task will take $O(\log n)$ time to run to completion, as we will see. This ensures that the *Assign*( ), *Rescale*( ), *Reassign*( ), and *Compact*( ) processes, when underway, always see a tree $T_1$ with consistent weights at the different head nodes.

Recall that we also need to track the weight of insertions made to each $i$-path so as to know when to initiate an $i$-process (it suffices to keep track of this value mod $2^{i+1}$). To this end, each update weight task also increments these weights. The update weight task for node $v$ needs to store two values: the first value is the increment being made to each of its ancestral centroid paths if it is underway, which equals the weight increment to $v$ between the start of the previous and current runs of the task; the second value is the weight increment to $v$ since the task last began running.

The weight update tasks are scheduled using the standard Dietz–Sleator cup-filling scheduler. A task's priority is given by the sum of the two increment values it holds. Here $atom = \Theta(p\text{-}inc) = \Theta(1/\log^2 n)$ and $k = \Theta(n/\log^3 n)$, for $p\text{-}inc$, the increase in priority during the execution of one task is defined to be a tight upper bound on the total weight increase to $T_1$ when performing one run of one task. We choose the constant of proportionality so that the start of successive runs of the task for a node $v$ are separated by at most a weight increase of $\alpha/\log n$ on $v$'s part, for a suitable constant $\alpha > 0$.

REMARK 6.23. *This implies that when a weight is updated, it is updated by at most* $p\text{-}inc \le \alpha/(4\log^2 n) \le 1/\log^2 n$, *as we will choose* $\alpha \le 1$.

By Corollary 6.22 this entails that $(9 + 4\log n)\, p\text{-}inc \le \alpha/\log n$, i.e., that an update task runs to completion during a period bounded by a weight increase of $\alpha/[(9 + 4\log n)\log n]$ to $T_1$. But such a weight increase requires $\Omega(\log n)$ insertions, and as one run of the task takes $O(\log n)$ time, this takes $O(1)$ time per insertion to $T$.

Now we can show the following lemma.

LEMMA 6.24. *The recorded size of an $i$-path is at most* $\alpha(2^{i+1}+2^{i-1}+2^{i-2})/\log n$ *smaller than the actual size and no larger than the actual size, assuming the actual size is less than* $2^{i+1} + 2^{i-1} + 2^{i-2}$.

*Proof.* Consider the subtree rooted at the head node of the $i$-path. If it has $r$ nodes of weight 1 it has at most $2r + 1$ nodes of weight less than 1 (since all but possibly one node of weight less than 1 has a parent, necessarily of weight 1, and since each weight 1 node has at most one child of weight less than 1). Since the subtree has size less than $2^{i+1} + 2^{i-1} + 2^{i-2}$, it has at most $2^{i+1} + 2^{i-1} + 2^{i-2}$ nodes of weight less than 1. But the recorded weight of each such node is in deficit by at most $\alpha/\log n$, and the result follows. ∎

We are ready to describe the scheduling of the *Assign*( ), *Rescale*( ), *Reassign*( ), and *Compact*( ) processes. For each $i$, we run a separate modified cup-filling procedure for the $i$-processes. The priority of a process is simply the weight of insertions in the associated subtree since the moment when the process would have been initiated in our original algorithm. For an *Assign*($\pi$) process this is approximated using the current size of $i$-path $\pi$ minus $2^i$; the size of $\pi$ when *Assign*($\pi$) should have been initiated. For *Compact*($\pi$) and *Reassign*( ) superprocesses, we need to record the weight of insertions mod $2^{i+1}$ that have occurred in the subtree rooted at the head of $i$-path $\pi$ since $\pi$ was created. This term minus the starting time of the process mod $2^{i+1}$, as specified in Invariant 1, yields the priority (for *Reassign*( ), the priority is calculated with a shift of $2^{i-3}$ mod $2^{i+1}$). It follows the recorded priority maybe too small, but by at most $\alpha(2^{i+1} + 2^{i-1} + 2^{i-2})/\log n$.

We perform each scheduled process for one basic step, cycling among the classes of processes for each class of $i$-path ($i = 0, 1, 2, \ldots$) in round robin order. We alternate between performing one basic step and one complete task updating path sizes. This ensures the recorded sizes of the centroid paths are always consistent when basic steps are being performed. Thus every $\Theta(\log^2 n)$ insertions, one basic step is performed on one process associated with a class $i$-path for each $i = 1, 2, \ldots$.

LEMMA 6.25. *With the following parameter choices, each $i$-process finishes within the time for $2^i/8$ weighted insertions into the corresponding $i$-path, assuming $n \geq 2$. The parameter choices are: error $= 11\,\alpha 2^{i-1}/\log n$, $\ell = e' \cdot i \cdot 2^i$ (measured in basic steps), $\lambda = d \log n$, p-inc $=$ atom$/(4d\log n)$, atom $= 44\alpha\ d\ 2^{i-1}$, $d = 2/\alpha$, $e' = \lceil \frac{e}{44} \rceil \cdot 44$, and $\alpha = 1/[4(154 + e')]$.*

*Proof.* We note that for these parameter values, $\ell$ is an integer multiple of *atom*, *atom* $\geq 4\lambda \max\{$*error*, *p-inc*$\}$, so Lemma 6.20 applies. Thus the process priorities are always bounded by $11\alpha\ 2^{i-1}/\log n + 11\alpha\ 2^{i-1}/\log n + 1/(d\log n)(44\alpha\ d\ 2^{i-1} + e' \cdot i \cdot 2^i) + 4(11\alpha\ 2^{i-1}/\log n + 11\alpha\ 2^{i-1}/\log n)\log n \leq \alpha\ 2^{i-1}(66/\log n + e' \cdot i/\log n + 88) \leq 2^i/[8 \cdot 4\,(154 + e')\,\alpha] = 2^i/8$, assuming $n \geq 2$. □

It remains to show that $O(1)$ work per weight increase of $4/\log^3 n$ suffices.

LEMMA 6.26. *It suffices to perform $O(1)$ work per weight increase of $4/\log^3 n$ to ensure that in the period in which an $i$-process performs atom basic steps the overall weight and hence priority increase by at most p-inc.*

*Proof.* These atom basic steps are performed over a period of $\Theta($*atom* $\log^2 n)$ insertions assuming $O(1)$ work per insertion. (Recall that we cycle among the $i$-processes for $i = 0, 1, \ldots, \log n$ in round robin order, performing one basic step on an $i$-process for each $i$, and each basic step takes $O(\log n)$ work.) Since each insertion causes a weight increase of at most $4/\log^3 n$, the resulting weight gain is $O($*atom*$/\log n)$. Notice that the more work per insertion, the fewer insertions needed to complete the *atom* basic steps and the smaller the weight increase. Thus with a large enough $O(1)$ work per insertion, the result holds. □

We have shown the following theorem.

THEOREM 6.27. *The implicit tree $T_1$ described above can be maintained in $O(1)$ work per insertion, assuming that each insertion results in a weight increase of $0$ or $4/\log^3 n$, that each insertion adds at most one weight $0$ node $v$ to $T_1$, and further that such a node $v$ is adjacent only to weight $1$ nodes. Further, LCA queries on $T_1$ can be answered in $O(1)$ time.*

## 7. The $O(1)$ worst-case algorithm.

The tree $T$ on which LCA queries are being performed is made binary, using a standard binarization. More specifically, a node $v$ with $d$ children is represented by $d$ copies of $v$ forming a chain of right children. The actual children of $v$ will be stored as the left children of this chain of nodes. Note that if $n$ items are inserted in an initially empty tree the binarized tree will contain at least $n$ nodes and at most $2n - 1$. As a result, an insertion may entail the addition of two nodes to the binarized tree, called $T$ henceforth. To simplify the discussion, from now on we term a node addition to $T$ an insertion, understanding that a real insertion may induce two insertions into $T$.

As already noted, $T$ is kept partitioned in at most $4n/\log^3 n$ subtrees, called partitioning subtrees, each of size at most $\log^3 n/4$ (strictly, $\lfloor (\log n)/4 \rfloor^3$). We assume that $n$ lies in the range $[n, 2n)$; section 8 explains how to handle $n$ increasing to $2n$ or beyond. We create a tree $T_1$ which contains the root of each subtree in the partition of $T$. These subtrees are chosen so that the LCA of the roots of any two partitioning subtrees is itself the root of a partitioning subtree. A node $v$ in $T_1$ is the parent of node $w$ in $T_1$ if $v$ is the nearest ancestor of $w$ in $T$ such that $v$ is in $T_1$. Clearly, as $T$ is binary, so is $T_1$.

When a partitioning subtree grows too large it is split, causing the addition of one or two nodes to $T_1$ (two nodes may be needed to maintain the LCA property on subtree roots). But, as we will see, a newly created partitioning subtree once initiated is itself partitioned only following $\Theta(\log^3 n)$ insertions into itself.

A partition of partitioning subtree $S$, rooted at node $v$, proceeds as follows. Once initiated, within $\frac{1}{4}\log^3 n$ further insertions into $S$ it determines the root(s) of the new subtrees. It then inserts one of the news roots in $T_1$ as a child of $v$, giving it weight 0. Over the next $\frac{1}{4}\log^3 n$ insertions into $S$ the weight of the new root is increased in increments of $4/\log^3 n$ until its actual weight is 1. Within the next $\frac{1}{8}\log^3 n$ insertions into $S$, we ensure $v$'s recorded weight becomes 1, as follows. Instead of following the previously stated rule for giving priorities to weight update tasks, once the actual weight of a node reaches 1, on each insertion to subtree $S$, we continue incrementing its priority by $4/\log^3 n$. To bring the new root's recorded weight to 1 may need the completion of one run of its weight increase task and a full second run of the task. We have ensured this occurs within a weight increase of $2\alpha/\log n$, so it suffices that $2\alpha/\log n \leq \frac{1}{8}\log^3 n \cdot 4/\log^3 n$, and $\alpha \leq 1/4$ suffices for $n \geq 2$. The second new root, if any, is then inserted in the same way. Note that this ensures that any node in $T_1$ of weight less than 1 is adjacent only to nodes of weight 1, and nodes of weight 1 have at most one child of weight less than 1.

To answer a query LCA$(u, v)$, we first determined if $u$ and $v$ are in different partitioning subtrees by finding, in $O(1)$ time, the roots $r_u$ and $r_v$ of their respective partitioning subtrees. If $r_u \neq r_v$, we compute LCA$(r_u, r_v)$ on $T_1$ in $O(1)$ time as previously described (see Theorem 6.27). Otherwise, the query is handled recursively.

To support queries on the partitioning subtrees, they are partitioned in turn into subtrees of size at most $4\log\log^3 n$.[8] For each partitioning subtree $S$ of $T$ we maintain a tree $S_1$ comprising the roots of $S$'s partitioning subtrees. Updates are performed using our previous algorithm, i.e., with $O(\log\log^3 n)$ work over $\Theta(\log\log^3 n)$ insertions. Queries are performed as in the previous paragraph. It is helpful to let $T_2$ denote the union of all the $S_1$ trees.

The recursion bottoms out at the partitioning subtrees of size $O(\log\log^3 n)$ for, as we will see, there are $o(n)$ distinct partitioning trees of this size, and their updating can be done via table lookup in $O(1)$ time per insertion, as can LCA queries. The requisite tables use $o(n)$ space.

## 7.1. Details of the data structures.

Each node in $T$ keeps a pointer to its newest ancestor in $T_2$, the root of the size $O(\log\log^3 n)$ partitioning subtree to which it belongs. Similarly, each node in $T_2$ keeps a pointer to its nearest ancestor in $T_1$, the root of the size $O(\log^3 n)$ partitioning subtrees to which it belongs. On an insertion, the weight of the appropriate nodes in $T_2$ and $T_1$ are incremented in $O(1)$ time, using the above pointers.

Definition 7.1. *The size of a partitioning subtree is the sum of the weights of the nodes it contains.*

The size of partitioning subtrees are recorded with their roots. On an insertion, the up to two subtree sizes that change are incremented (by $4/\log\log^3 n$ and $4/\log^3 n$, respectively); these sizes are stored at the subtrees' roots.

Additional data is needed to support the splitting of the partitioning subtrees. We begin by describing what is needed for splitting the size $O(\log\log^3 n)$ partitioning subtrees. In addition to storing the subtrees themselves, we keep a table of all possible trees, represented canonically. Using the canonical representation, in $O(1)$ time we will be able to answer LCA queries and to determine the new canonical tree resulting

[8] $\log\log^3 n$ is our notation for $(\log\log n)^3$.

from an insertion. Finally, by linking the nodes of the actual tree to those of the corresponding canonical tree, we will be able to translate query answers on canonical tree to answers on the actual tree in $O(1)$ time.

The following information is maintained for each actual tree $S$.

1. For each node $v$ in $S$, a distinct label, denoted $label(v)$ in the range $[1, \log\log^3 n]$. In addition, the up to two edges going to children outside $S$ are also recorded. (The structure of $S$ along with the associated labels provides the appropriate canonical labelled tree used to answer LCA queries on $S$.)
2. An array $\ell\_to\_n(S)$ storing, for each label in the range $[1, \log\log^3 n]$, the node in $S$ corresponding to this label, if any. This inverse map is used to convert the LCA obtained using lookup tables on the canonical tree from a label to an actual node (for the canonical tree nodes are named by their labels).
3. The name $name(S)$ of the labelled canonical tree associated with $S$, the root $root(S)$, of $S$, along with a pointer $pointer(v)$ from each node $v$ in $S$ to the location storing $\ell\_to\_n(S)$, $name(S)$, $root(S)$, $size(S)$, and $flag(S)$. The role of $flag(S)$ is explained next.
4. Actually, two copies of $label(v)$ and $pointer(v)$ are maintained for each node $v$ in $S$. One of these copies will be "old" and the other "current." This will be indicated by the $flag(S)$ field above. The $flag(S)$ field pointed to by the "old" $pointer(v)$ will be set to 0 while that pointed to by the "current" $pointer(v)$ will be set to 1.
5. In addition to the above, there is a static table for each labelled tree of size at most $4\log\log^3 n$ supporting the following queries: given two labels in the tree, return the label of the LCA, and given a new label (corresponding to a newly inserted node) and the label(s) corresponding to the node(s) at the insertion site, return the name of the resulting labelled tree. Note that labels for nodes in $S$ are allocated in sequential order of insertion.

   Since there are $O(2^{8\log\log^3 n}(4\log\log^3 n)^{4\log\log^3 n})$ labelled binary trees of size at most $4\log\log^3 n$ with labels in the range $[1, \log\log^3 n]$, the total space occupied by the above tables is $O(n)$. These tables can also be built in $O(n)$ time.

*Processing insertions.* Each insertion will do $O(1)$ work at each of the 3 levels. This will result in $\Theta(\log^3 n)$ work being available for each insertion into $T_1$ and $\Theta(\log\log^3 n)$ work for each insertion into $T_2$.

Insertions into $T$ will require the following actions. First, the insertion into the appropriate size $O(\log\log^3 n)$ partitioning subtree $S$ of $T$ rooted at a node in $T_2$ is made. This is done using a constant time table lookup to calculate the name of the new subtree after insertion. Second, if $S$ reaches size $3(\log\log^3 n)$ then it is partitioned into two or three subtrees, each of size at most $3\log\log^3 n$, over the next $\log\log^3 n$ insertions to $S$.

An insertion of node $u$ into $T$ is processed as follows.

Let $v$ be the parent of $u$ in $T$. $u$ is viewed as being inserted into the partitioning subtrees $S_b$ and $S_a$ containing $v$, of sizes $O(\log\log^3 n)$ and $O(\log^3 n)$ and rooted in $T_2$ and $T_1$, respectively. On following $pointer(v)$, $\ell\_to\_n(S_b)$, $name(S_b)$, and $size(S_b)$ are readily updated in $O(1)$ time (using table lookup for $name(S_b)$). If $size(S_b)$ reaches $3\log\log^3 n$, a split of $S_b$ is initiated. It is carried out as described below. $O(1)$ work is then performed on the tasks associated with the trees $T_1$ and $T_2$.

*Splitting $S = S_b$.* The first step is to find a splitting location that divides the tree into two pieces each of size at least $\log\log^3 n$. This can be done by depth first search in $O(\log\log^3 n)$ time, or by table lookup in $O(1)$ time. To ensure that the new trees have at most two external children each, we find the LCAs of the new roots and the up to two external children; if one of these LCAs is not a new root, it is also introduced as a third root. The one or two new roots are added to tree $T_2$ with the already explained timing (it suffices to carry out the depth first search within $\frac{1}{4}\log\log^3 n$ insertions). To simplify the notation, we continue to suppose that only two new trees are created; the changes if there are three new trees are evident.

The new roots define $S_1$ and $S_2$, the trees that $S_b$ is split into. Next, $\mathit{root}(S_1)$, $\ell\_to\_n(S_1)$, $\mathit{name}(S_1)$, $\mathit{size}(S_1)$, $\mathit{root}(S_2)$, $\ell\_to\_n(S_2)$, $\mathit{size}(S_2)$, and $\mathit{name}(S_2)$ are computed in the obvious way in $O(\log\log^3 n)$ time (e.g., by traversing each of $S_1$ and $S_2$ in turn and "inserting" their nodes one by one). Then for each node $w$ the "old" $\mathit{label}(w)$ and $\mathit{pointer}(w)$ are updated to be in accordance with $S_1$ or $S_2$, whichever contains $v$, also in $O(\log\log^3 n)$ time.

Note that all this while the "current" $\mathit{label}(v)$, $\mathit{pointer}(v)$, $\mathit{name}(S)$, $\mathit{root}(S)$, and $\ell\_to\_n(S)$ are used to answer LCA queries; furthermore these structures are updated with each insertion that occurs even after the splitting process starts. Also note that after the splitting process starts, new insertions are neglected in constructing $S_1$ and $S_2$ and the associated fields $\mathit{name}(S_1)$, $\mathit{root}(S_1)$, $\ell\_to\_n(S_1)$, $\mathit{name}(S_2)$, $\mathit{root}(S_2)$, $\ell\_to\_n(S_2)$. This is easily implemented by putting a time-stamp on each inserted node and ignoring nodes which are time-stamped later than the start of the splitting process. These insertions are just queued up and performed on $S_1$ or $S_2$ as appropriate after they have been constructed. When all $\log\log^3 n$ insertions have been performed, $\mathit{flag}(S_1)$ and $\mathit{flag}(S_2)$ are set and $\mathit{flag}(S)$ is reset so that for each $v$ in $S_1$ and $S_2$ the "old" $\mathit{label}(v)$ and $\mathit{pointer}(v)$ become "current" and vice versa; this takes $O(1)$ time. Note that $S_1$ and $S_2$ each have size at most $3\log\log^3 n$ at this point.

*Splitting algorithm for a size $O(\log^3 n)$ partitioning subtree $S_a$.* The splitting algorithm on $S_a$ begins when its size reaches $3\log^3 n$ and is similar to the previous splitting algorithm but is done without table lookup. For each partitioning $S_a$ the following information is maintained, in addition to the data structure storing the tree itself.

1. For each node $v$, $\mathit{code}(v)$, along with the annotations, and $\mathit{number}(v)$.
2. The structure $\mathit{cpath}(S_a)$ storing centroid paths and associated information for paths in $S_a$.
3. For each node $v$, a pointer $\mathit{pointer}(v)$ to the location storing $\mathit{cpath}(S_a)$, $\mathit{root}(S_a)$, and $\mathit{flag}(S_a)$.
4. Two copies of $\mathit{code}(v)$, $\mathit{number}(v)$, $\mathit{pointer}(v)$, maintained as before. One of these copies will be "old" and the other "current." The appropriate $\mathit{flag}(S_a)$ indicates which of these copies is current.

The algorithm proceeds as before. First, the location of the split is determined in $O(\log^3 n)$ time using depth-first search. This splits $S_a$ into two pieces each of size at least $\log^3 n$, thereby defining $S_1$ and $S_2$. Then the data structures for $S_1$ and $S_2$ are computed, the backlog of insertions is applied, and finally the appropriate flats are set. This all takes $O(\log^3 n)$ time.

We have shown the following lemma.

LEMMA 7.2. *There is a data structure for trees of size in the range $[n, 2n)$ which answers LCA queries in $O(1)$ time and performs insertions to the tree in $O(1)$ time.*

REMARK 7.3. *Although space is not freed when subtrees are split, the space used is still linear. For each split of a size $s$ tree, using space $\Theta(s)$, only happens after $\Theta(s)$ insertions.*

## 8. Handling deletions and changing values of $n$.

We note that the need for the limited range in Lemma 7.2 arises for two reasons: first, the construction in section 7 requires a fixed value of $\log^3 n$ (and of $\log\log^3 n$), and second, the basic algorithm takes $O(\log^3 n)$ time per update; here the $\log^3 n$ is not fixed. But we could readily change the range to $[n, 2^d n)$ for any fixed $n$ by replacing $\log^3 n$ by $(\log n + d)^3$ in section 7. As we will see, $d = 3$ suffices.

We do not perform deletions explicitly. Instead, deleted items will just be marked as such, or rather the topmost copy of the corresponding node will be so marked. Thus the size of the current tree would include the count of both deleted and nondeleted nodes. We will periodically rebuild the data structure from scratch, in the background, so as to maintain the following invariant.

*Invariant* 3. The insertion count, the number of items in the data structure plus the number of items marked deleted, lies in the range $[4\cdot 2^i, 32\cdot 2^i]$; the actual number of items in the data structure lies in the range $[6\cdot 2^i, 32\cdot 2^i]$. In addition, the number of deleted items is at most $3\cdot 2^i$. Furthermore, immediately after being rebuilt the insertion count is in the range $[8\cdot 2^i, 31\cdot 2^i]$, the actual count in the range $[7\cdot 2^i, 31\cdot 2^i]$, and the deletion count is at most $2^{i+1}$.

Invariant 3 implies that the number of nodes in the tree lies in the range $[4\cdot 2^i, 64\cdot 2^i)$, and thus $d = \log 64/4 = 4$ suffices.

Let the actual size of the data structure denote the number of undeleted items. If the insertion count reaches $31\cdot 2^i$, the data structure is rebuilt in the range $[8\cdot 2^i, 64\cdot 2^i]$; if the actual size decreases to $7n$, then it is rebuilt in the range $[2\cdot 2^i, 16\cdot 2^i]$; if neither of these apply but the number of items marked deleted reaches $2^{i+1}$, then it is rebuilt either in the range $[4\cdot 2^i, 32\cdot 2^i]$ if the actual size is at least $8n$, and in the range $[2\cdot 2^i, 16\cdot 2^i]$ otherwise. The rebuilding is completed within $2^i$ insertions; this takes $O(1)$ time per insertion. It is readily checked that Invariant 3 continues to hold following the rebuilding.

An easy way to perform the rebuilding is to traverse the current tree, determining the undeleted items and then inserting them in the new tree, one by one, using the appropriate value for $\log^3 n$. Of course, new insertions and deletions are performed on the current tree and queued so they can be performed on the new tree.

We have shown the following theorem.

THEOREM 8.1. *There is a linear time data structure that supports LCA queries on a tree undergoing insertions and deletions such that each update and query can be performed in worst case $O(1)$ time.*

**Acknowledgments.** We thank the referees for urging us to seek a less complex solution which led us to the present version of the data structure.

## REFERENCES

[BF00] M. BENDER AND M. FARACH-COLTON, *The LCA problem revisited*, in Proceedings of Latin American Theoretical Informatics, 2000, pp. 88–94.

[BV94] O. BERKMAN AND U. VISHKIN, *Finding level ancestors in trees*, J. Comput. System Sci., 48 (1994), pp. 214–230.

[CH97] R. COLE AND R. HARIHARAN, *Approximate string matching: A simpler faster algorithm*, SIAM J. Comput., 31 (2002), pp. 1761–1782.

[DS87] P. DIETZ AND D. SLEATOR, *Two algorithms for maintaining order in a list*, in Proceedings of the 19th ACM Symposium on Theory of Computing, 1987, pp. 365–371.

[F99] M. FARACH-COLTON, *private communication*, 1999.

[Ga90] H. Gabow, *Data structures for weighted matching and nearest common ancestors with linking*, in Proceedings of the 1st ACM-SIAM Symposium on Discrete Algorithms, 1990, pp. 434–443.

[Gus97] D. Gusfield, *Algorithms on Strings, Trees, and Sequences: Computer Science and Computational Biology*, Cambridge University Press, Cambridge, UK, 1997, pp. 196–207.

[HT84] D. Harel and R. Tarjan, *Fast algorithms for finding nearest common ancestors*, SIAM J. Comput., 13 (1984), pp. 338–355.

[LV86] G. Landau and U. Vishkin, *Fast parallel and serial approximate string matching*, J. Algorithms, 10 (1989), pp. 157–169.

[M76] E. McCreight, *A space-economical suffix tree construction algorithm*, J. ACM, 23 (1976), pp. 262–272.

[Meh77] K. Mehlhorn, *A best possible bound for the weighted path length of binary search trees*, SIAM J. Comput., 6 (1977), pp. 235–239.

[SV88] B. Schieber and U. Vishkin, *On finding lowest common ancestors: Simplification and parallelization*, SIAM J. Comput., 17 (1988), pp. 1253–1262.

[Ts84] A. Tsakalidis, *Maintaining order in a generalized link list*, Acta Inform., 21 (1984), pp. 101–112.

[We92] J. Westbrook, *Fast incremental planarity searching*, in Proceedings of the 19th International Colloquium on Automata, Languages and Programming, Lecture Notes in Comput. Sci. 623, Springer-Verlag, 1992, pp. 342–353.

[W82] D. Willard, *Maintaining dense sequential files in a dynamic environment*, in Proceedings of the 24th ACM Symposium on Theory of Computing, 1982, pp. 114–121.

SIAM J. Comput. © 2005 Society for Industrial and Applied Mathematics
Vol. 34, No. 4, pp. 924–945

# ORDERLY SPANNING TREES WITH APPLICATIONS*

YI-TING CHIANG†, CHING-CHI LIN†, AND HSUEH-I LU‡

**Abstract.** We introduce and study *orderly spanning trees* of plane graphs. This algorithmic tool generalizes *canonical orderings*, which exist only for triconnected plane graphs. Although not every plane graph admits an orderly spanning tree, we provide an algorithm to compute an *orderly pair* for any connected planar graph $G$, consisting of an embedded planar graph $H$ isomorphic to $G$, and an orderly spanning tree of $H$. We also present several applications of orderly spanning trees: (1) a new constructive proof for Schnyder's realizer theorem, (2) the first algorithm for computing an area-optimal 2-visibility drawing of a planar graph, and (3) the most compact known encoding of a planar graph with $O(1)$-time query support. All algorithms in this paper run in linear time.

**Key words.** planar graph algorithm, graph drawing, realizer, visibility representation, canonical ordering, orderly spanning tree, graph encoding, triangulation, unit-cost RAM model, succinct data structure, data compression

**AMS subject classifications.** 05C62, 05C85, 68P05, 68W35, 68U05, 68R10, 94C15

**DOI.** 10.1137/S0097539702411381

## 1. Introduction.

A plane graph is a planar graph equipped with a plane embedding. *Canonical orderings* of triconnected plane graphs [11, 19, 27, 28] are crucial in several graph-drawing and graph-encoding algorithms [7, 8, 9, 16, 20, 22]. This paper introduces the *orderly spanning tree* as an algorithmic tool that generalizes the concept of canonical orderings for plane graphs that are not required to be triconnected. The concept of orderly spanning trees of plane graphs originates from that of canonical spanning trees of triconnected plane graphs [9], but the former is more general even for triconnected plane graphs (see section 2.1 and Figure 2.1(b)).

We say that $(H, T)$ is an *orderly pair* of $G$ if (1) $T$ is an orderly spanning tree of plane graph $H$, and (2) $G$ and $H$, ignoring their embeddings, are isomorphic planar graphs. Although not every connected plane graph admits an orderly spanning tree (see section 2.1 and Figure 2.2(a)), we provide a linear-time algorithm (see section 2.2) to compute an orderly pair for any connected planar or plane graph. We also present three applications of orderly spanning trees in the paper.

*Application* 1: *Realizers of planar graphs.* A graph is *simple* if it contains no multiple edges. For the first application of orderly spanning trees, we present a new linear-time algorithm to compute a *realizer* for any plane triangulation (i.e., simple triangulated plane graph with at least three nodes). Schnyder [38] gave the first known linear-time algorithm that computes a realizer for any plane triangulation, thereby settling the open question on the dimension [42, 14] of planar graphs. This celebrated result also yields the best known straight-line drawing of planar graphs on

*Received by the editors July 13, 2002; accepted for publication (in revised form) October 29, 2004; published electronically May 12, 2005. A preliminary version appeared in *Proceedings of the 12th Annual ACM–SIAM Symposium on Discrete Algorithms*, Washington, D.C., 2001, pp. 506–515. This research is supported in part by NSC grants 89-2213-E-001-034, 89-2218-E-001-014, and 93-2213-E-001-002.

http://www.siam.org/journals/sicomp/34-4/41138.html

†Institute of Information Science, Academia Sinica, 128 Academia Road, Section 2, Taipei 115, Taiwan, Republic of China.

‡Department of Computer Science and Information Engineering, National Taiwan University, 1 Roosevelt Road, Section 4, Taipei 106, Taiwan, Republic of China (hil@csie.ntu.edu.tw, http://www.csie.ntu.edu.tw/~hil/).

the grid [39]. Our proof, based upon the existence of an orderly spanning tree for any simple plane triangulation, is quite simple.

*Application* 2: *Optimal* 2*-visibility drawings of planar graphs.* For the second application of orderly spanning trees, we give an $O(n)$-time algorithm that produces a 2-visibility drawing of area at most $(n-1) \times \lfloor \frac{2n+1}{3} \rfloor$ for any $n$-node simple plane graph $H$ with $n \geq 3$. Let $v_1, v_2, \ldots, v_n$ be the nodes of $H$. A 2*-visibility drawing* [16] of $H$ consists of $n$ nonoverlapping rectangles $b_1, b_2, \ldots, b_n$ such that if $v_i$ and $v_j$ are adjacent in $H$, then $b_i$ and $b_j$ are visible to each other either horizontally or vertically.[1] For example, the picture in Figure 1.1(b) is a 2-visibility drawing of the plane graph in Figure 1.1(a). Fößmeier, Kant, and Kaufmann [16] gave an $O(n)$-time algorithm to compute an $x \times y$ 2-visibility drawing for $H$ with $x + y \leq 2n$ and conjectured that it is "not trivial" to improve their upper bound. Moreover, they showed an $n$-node plane triangulation whose $x \times y$ 2-visibility drawing requires $x + y \geq n - 1 + \lfloor \frac{2n+1}{3} \rfloor$ and $\min\{x, y\} \geq \lfloor \frac{2n+1}{3} \rfloor$.[2] According to their lower bounds, the 2-visibility drawing produced by our algorithm is worst-case optimal.

In order to take advantage of the wonderful properties of canonical orderings, many drawing algorithms work on triangulated versions of input plane graphs. As pointed out in [19], the initial triangulation tends to ruin the original plane graph's structure. Our orderly pair algorithm appears as a promising tool for drawing graphs neatly and compactly, without first triangulating the given plane graphs. The concept of an orderly pair is more general than that of a canonical ordering, since all known canonical orderings are only defined for triconnected plane graphs. The technique of orderly pairs is potentially more powerful, since it exploits the flexibility of planar graphs whose planar embeddings are not predetermined.

*Application* 3: *Convenient encodings of planar graphs.* For the third application of orderly spanning trees, we investigate the problem of encoding a graph $G$ into a binary string $S$ with the requirement that $S$ can be decoded to reconstruct $G$. This problem has been extensively studied with three objectives: (1) minimizing the length of $S$, (2) minimizing the time required to compute and decode $S$, and (3) supporting queries efficiently. As these objectives are often conflicting, a number of coding schemes with different trade-offs have been proposed in the literature. The widely useful adjacency-list encoding of an $n$-node $m$-edge graph $G$ requires $2m\lceil \log_2 n \rceil$ bits. See [9, 22, 23, 29, 34, 37, 43] for $O(n)$-bit encodings of $n$-node planar graphs without efficient query supports.

Under the model of unit-cost RAM [5, 10, 17, 40, 41, 45], where operations such as read, write, and add on $O(\log n)$ consecutive bits take $O(1)$ time, an encoding $S$ of $G$ is *weakly convenient* [9] if it takes (i) $O(m+n)$ time to encode $G$ and decode $S$, (ii) $O(1)$ time to determine from $S$ the adjacency of any two nodes in $G$, and (iii) $O(d)$ time to determine from $S$ the neighbors of a degree-$d$ node in $G$. If the degree of a node can be determined from a weakly convenient encoding $S$ in $O(1)$ time, then $S$ is *convenient* [9]. For a planar graph $G$ having multiple edges but no self-loops, Munro and Raman [35] gave the first nontrivial convenient encoding of $G$ with $2m + 8n + o(m+n)$ bits. Their result is based on the four-page decomposition of planar graphs [46] and auxiliary strings, encoding an involved three-level data

[1]A closely related *rectangle-visibility drawing* [13, 12, 25, 3] of $H$ requires that $v_i$ and $v_j$ are adjacent in $H$ *if and only if* $b_i$ and $b_j$ are visible to each other.

[2]The lower bounds stated in [16] are $x + y \geq \frac{5n}{3}$ and $\min\{x, y\} \geq \frac{2n}{3}$. Based on the given sketch of proof, however, it is not hard to see that their lower bound should be corrected as $x + y \geq n - 1 + \lfloor \frac{2n+1}{3} \rfloor$ and $\min\{x, y\} \geq \lfloor \frac{2n+1}{3} \rfloor$.

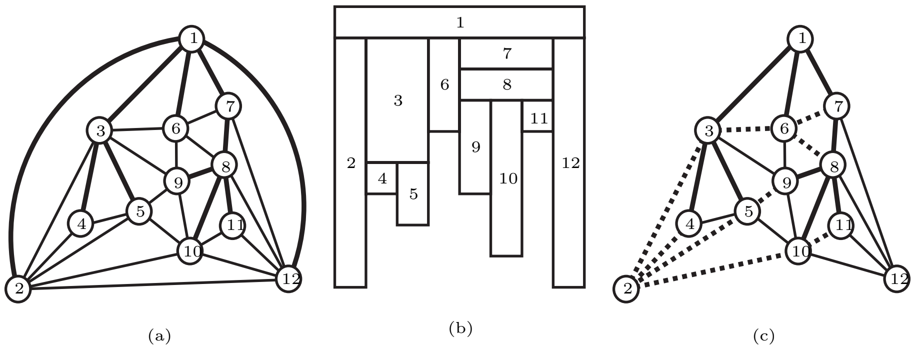


Fig. 1.1. (a) *A plane graph $H$ with an orderly spanning tree of $H$ rooted at node* 1 *represented by the thick edges.* (b) *A* 2*-visibility drawing of $H$.* (c) *A realizer $(T_1, T_2, T_{12})$ of $H$, where $T_1$* (*respectively, $T_2$ and $T_{12}$*) *consists of the thick* (*respectively, dashed and thin*) *edges.*

structure for any string of parentheses. For a planar graph $G$ that has (respectively, has no) multiple edges, Chuang, Garg, He, Kao, and Lu [9] improved the bit count to $2m + \left(5 + \frac{1}{k}\right)n + o(m+n)$ (respectively, $\frac{5}{3}m + \left(5 + \frac{1}{k}\right)n + o(n)$) for any positive constant $k$. They also provided a weakly convenient encoding of $2m + \frac{14}{3}n + o(m+n)$ (respectively, $\frac{4}{3}m + 5n + o(n)$) bits for a planar graph $G$ that has (respectively, has no) multiple edges. Based on our orderly pair algorithm, in this paper we present the best known convenient encodings for a planar graph $G$: If $G$ may (respectively, does not) contain multiple edges, then the bit count of our encoding is $2m + 3n + o(m+n)$ (respectively, $2m + 2n + o(n)$), which is even less than that of the weakly convenient encodings of Chuang et al. [9]. The bit counts are very close to Tutte's information-theoretical lower bound of roughly $3.58m$ bits for encoding connected plane graphs without any query support [44]. The bit count of our encoding for a planar graph $G$ without multiple edges matches that of the best known convenient encoding for an outerplanar graph [35]. Besides relying on the orderly pair algorithm, our results are also based on an improved auxiliary string for a folklore encoding [9, 21, 35] of a rooted tree $T$. With the auxiliary strings of Munro and Raman [35], computing the degree of a degree-$d$ node in $T$ requires $\Theta(d)$ time. In this paper, we present a nontrivial auxiliary string, in Lemma 5.3, to support the degree query in $O(1)$ time.

*Recent applications.* Besides the applications presented in the present paper, our orderly pair algorithm also yields the following recent results:

- Improved compact distributed routing tables for any $n$-node distributed planar network [33], improving the best previously known design of Gavoille and Hanusse [18] by reducing the worst-case table size count from $8n + o(n)$ bits to $7.181n + o(n)$ bits, without increasing the time complexity of preprocessing and query.
- A linear-time algorithm for compact floor-plans of plane triangulations [30, 31], which is not only much simpler than the previous methods in the literature [20, 47] but also provides the first known nontrivial upper bound on the floor-plan's area.
- Compact Podevs drawings for plane graphs and an alternative proof for the sufficient and necessary condition for a planar graph to admit a rectangular dual [6].
- Improved upper bounds on the number of planar graphs via so-called *well orderly spanning trees*, which are orderly spanning trees with additional prop-

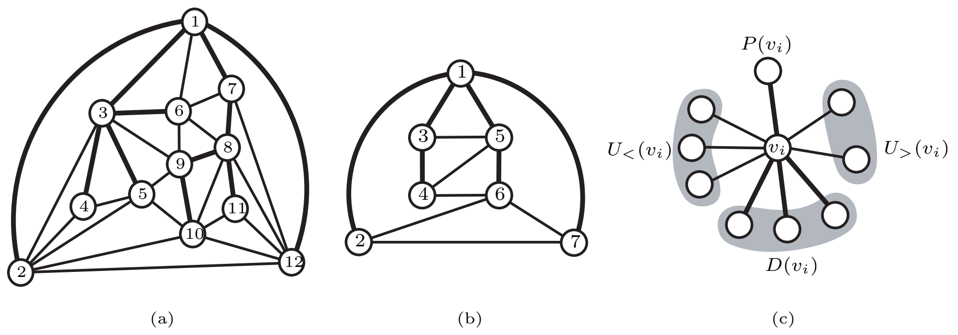


FIG. 2.1. (a) *The tree rooted at node* 1*, consisting of the thick edges, is not an orderly spanning tree of the plane graph. (b) A triconnected plane graph $H$, where the thick edges form an orderly spanning tree $T$, rooted at node* 1*, of $H$. The counterclockwise preordering of $T$ is not a canonical ordering of $H$.* (c) *Illustration for the orderly pattern of $v_i$.*

erties [2].

*Organization of the paper.* The rest of the paper is organized as follows. Section 2 gives the linear-time algorithm for computing an orderly pair of any given planar graph. Applications are given in sections 3–5. Section 3 gives the linear-time algorithm for computing a realizer of any given plane triangulation. Section 4 shows the linear-time algorithm for obtaining an area-optimal 2-visibility drawing of any given plane graph. Section 5 presents the best known convenient encodings for planar graphs.

## 2. Orderly spanning trees for plane graphs.

### 2.1. Basics.

Unless stated otherwise, all graphs in sections 2–4 are simple. Let $H$ be a plane graph. The *outer boundary* of $H$ is the boundary of the external face of $H$. The nodes and edges on the outer boundary of $H$ are *external* in $H$; and the other nodes and edges are *internal* in $H$.

Let $T$ be a rooted spanning tree of a connected plane graph $H$. Two distinct nodes of $H$ are *unrelated* with respect to $T$ if neither of them is an ancestor of the other in $T$. An edge $e$ of $H$ is *unrelated* with respect to $T$ if the endpoints of $e$ are unrelated with respect to $T$. Let $v_1, v_2, \ldots, v_n$ be the counterclockwise preordering of the nodes in $T$. A node $v_i$ is *orderly* in $H$ with respect to $T$ if the neighbors of $v_i$ in $H$ form the following four blocks of $H$ with respect to $T$ in counterclockwise order around $v_i$, where each block could be empty.

- $P(v_i)$: the parent of $v_i$ in $T$;
- $U_<(v_i)$: the nodes $v_j$ with $j < i$ that are unrelated to $v_i$ with respect to $T$;
- $D(v_i)$: the children of $v_i$ in $T$; and
- $U_>(v_i)$: the nodes $v_j$ with $j > i$ that are unrelated to $v_i$ with respect to $T$.

(See Figure 2.1(c).) $T$ is an *orderly spanning tree* of $H$ if (i) $v_1$ is on the outer boundary of $H$ and (ii) each node $v_i$ is orderly in $H$ with respect to $T$. If $T$ is an orderly spanning tree of $H$, then each incident edge of $v_1$ in $H$ belongs to $T$. An example of an orderly spanning tree is given in Figure 1.1(a). Figure 2.1(a) provides a negative example of an orderly spanning tree, where nodes 1, 3, 8, and 10 are not orderly in $H$ with respect to $T$.

Not every connected plane graph admits an orderly spanning tree. However, as to be shown in this section, there always exists a planar embedding for any given planar graph that admits an orderly spanning tree. For example, consider the plane graph

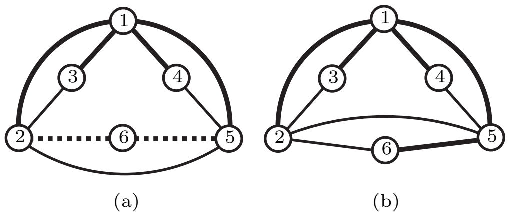


FIG. 2.2. (a) *A plane graph $H$ that has no orderly spanning trees.* (b) *A different planar embedding of $H$ that admits an orderly spanning tree rooted at node* 1*, consisting of the thick edges.*

$H$ in Figure 2.2(a). Assume for a contradiction that $H$ admits an orderly spanning tree $T$ rooted at node 1. Observe that the thick edges must be in $T$, and thus the thin edges cannot be in $T$. Now, $T$ contains exactly one of the dashed edges. In either case, however, the parent of node 6 in $T$ is not orderly in $H$ with respect to $T$, thereby contradicting the assumption that $T$ is an orderly spanning tree rooted at node 1. Since $H$ is rotationally symmetric, $H$ admits no orderly spanning trees. If we change the planar embedding of $H$ by moving edge $(2, 5)$ to the interior of $H$, as shown in Figure 2.2(b), then the new plane graph has an orderly spanning tree rooted at node 1 consisting of the thick edges.

The concept of orderly spanning trees of plane graphs originates from those of canonical orderings and canonical spanning trees of triconnected plane graphs. Let $H$ be a triconnected plane graph. A canonical ordering of $H$ is a certain ordering of the vertices in $H$, first introduced by de Fraysseix, Pach, and Pollack [11] for plane triangulations, and extended to triconnected plane graphs by Kant [27]. Specifically, let $v_1, v_2, \ldots, v_n$ be an ordering of the nodes of $H$. Let $H_i$ be the subgraph of $H$ induced by $v_1, v_2, \ldots, v_i$. Let $B_i$ be the outer boundary of $H_i$. This ordering is *canonical* if the interval $[3, n]$ can be partitioned into $I_1, \ldots, I_K$ with the following properties for each $I_j$. Suppose $I_j = [k, k + q]$. Let $C_j$ be the path $v_k, v_{k+1}, \ldots, v_{k+q}$.

- $H_{k+q}$ is biconnected. $B_{k+q}$ contains the edge $(v_1, v_2)$ and $C_j$. $C_j$ has no chord in $H$.
- If $q = 0$, $v_k$ has at least two neighbors in $H_{k-1}$, all on $B_{k-1}$. If $q > 0$, $C_j$ has exactly two neighbors in $H_{k-1}$, both on $B_{k-1}$, where the left neighbor is incident to $C_j$ only at $v_k$ and the right neighbor only at $v_{k+q}$.
- For each $v_i$ with $k \le i \le k + q$, if $i < n$, $v_i$ has at least one neighbor in $H - H_{k+q}$.

Chuang et al. [9] defined a canonical spanning tree $T$ of $H$ as a way to find parents in $T$ for all except one node of $H$ according to any given canonical ordering of $H$. Specifically, for the given canonical ordering of $H$, the *canonical spanning tree* $T$ of $H$ rooted at $v_1$ is the one formed by the edge $(v_1, v_2)$ together with the paths $C_j$ and the edges $(v_\ell, v_k)$ over all $I_j$, where $v_\ell$ is the leftmost neighbor of $C_j$ on $B_{k-1}$. By the triconnectivity of $H$, it is implicit in [9] that (a) any canonical spanning tree $T$ of $H$ has to be an orderly spanning tree of $H$, and (b) the counterclockwise preordering of $T$ is the given canonical ordering of $H$. As shown in Figure 2.1(b), however, the counterclockwise preordering of an orderly spanning tree for a triconnected plane graph $H$ may not be a canonical ordering of $H$. Therefore, the concept of orderly spanning trees is more general than that of canonical spanning trees even for triconnected plane graphs. The counterclockwise preordering of any orderly spanning tree of a plane triangulation $\Delta$ has to be a canonical ordering of $\Delta$, though.

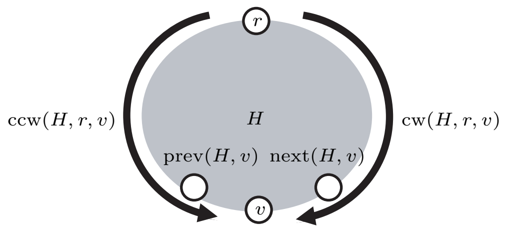


FIG. 2.3. *Illustration of the definitions of prev*$(H,v)$*, next*$(H,v)$*, cw*$(H,r,v)$*, and ccw*$(H,r,v)$*.*

**2.2. The orderly pair algorithm.** For plane graphs $H$ and $G$, $H \sim G$ denotes that $H$ and $G$ (ignoring their plane embeddings) are isomorphic planar graphs. We say that $(H,T)$ is an *orderly pair* of a connected planar graph $G$ with respect to $r$ if (i) $H \sim G$ and (ii) $T$ rooted at $r$ is an orderly spanning tree of $H$. This subsection shows how to compute an orderly pair for any planar graph in linear time. Without loss of generality, we may assume that the input planar graph is already equipped with a planar embedding represented by an adjacency list, where each node $v$ keeps a doubly linked list, storing its neighbors in counterclockwise order around $v$. Moreover, the two copies of each edge are cross-linked to each other. Such a representation can be obtained as a by-product by running the linear-time planarity-testing algorithm of Hopcroft and Tarjan [24] or that of Boyer and Myrvold [4]. Based upon this representation, deleting an edge takes $O(1)$ time. Moreover, moving an edge $e$ to the interior of a face $F$ can be conducted in $O(1)$ time, as long as an edge on the boundary of $F$ incident to each endpoint of $e$ is provided. (Our algorithm moves an edge $e$ to the interior of $F$ only when it traverses the boundary of $F$.)

To describe the algorithm, we need some definitions for a 2-connected plane graph $H$. If $v$ is an external node in $H$, then let $\mathrm{next}H, v$ (respectively, $\mathrm{prev}H, v)$ denote the external node of $H$ that immediately succeeds (respectively, precedes) $v$ in counterclockwise order around the outer boundary of $H$. For any two distinct external nodes $r$ and $v$ of $H$, let $\mathrm{ccw}(H,r,v)$ (respectively, $\mathrm{cw}(H,r,v)$) denote the sequence of the external nodes of $H$ from $r$ to $v$ in counterclockwise (respectively, clockwise) order around the outer boundary of $H$. Observe that $\mathrm{prev}(H,v) \in \mathrm{ccw}(H,r,v)$ and $\mathrm{next}(H,v) \in \mathrm{cw}(H,r,v)$. Define $\mathrm{boundary}(H,r) = \mathrm{ccw}(H,r,\mathrm{prev}(H,r))$, i.e., the sequence of the external nodes of $H$ from $r$ to $\mathrm{prev}(H,r)$ in counterclockwise order around the outer boundary of $H$. See Figure 2.3 for an illustration. For example, if $H$ is the plane graph shown in Figure 2.2(b), then we have that $\mathrm{next}(H,2) = 6$, $\mathrm{prev}(H,2) = 1$, $\mathrm{ccw}(H,1,6) = (1,2,6)$, $\mathrm{cw}(H,1,6) = (1,5,6)$, and $\mathrm{boundary}(H,1) = (1,2,6,5)$.

The key component of our orderly pair algorithm is the recursive subroutine $\mathsf{block}(G,r,v)$ shown below. Given any 2-connected plane graph $G$ and two distinct external nodes $r$ and $v$ of $G$, the subroutine $\mathsf{block}(G,r,v)$ computes an orderly pair $(H,T)$ of $G$ with respect to $r$. Let us emphasize that $v$ can be any external node of $G$ other than $r$. The high-level strategy of $\mathsf{block}(G,r,v)$ is to identify a neighbor $p$ of $v$ in $G$ that can be the parent of $v$ in $T$. The subroutine then deletes $v$ and its incident edges and recursively works on each 2-connected component of the remaining graph. The difficulty lies in efficiently choosing an appropriate parent of $v$ with possible modification to the plane embedding of $G$. More precisely, we want the subroutine to alter the embedding of $G$ into $H$ and find a neighbor $p$ of $v$ such that (1) the required time is linear in the total size of the internal faces of $G$ that contains $v$ and (2) the following *property* $\Pi$ holds for $H$ and $p$:

> For each neighbor $x$ of $v$ in $H$ other than $p$, if $x$ and $\mathrm{prev}(H, v)$ (respectively, $\mathrm{next}(H, v)$) are on the same side of $(v, p)$ in $H$, then $(v, x)$ is on the first (respectively, last) internal face of $H$ containing $v$ and $x$ in counterclockwise order around $v$ starting from the one containing $(v, \mathrm{next}(H, v))$.

(See Figure 2.5(b) for an illustration of property $\Pi$.) It turns out that a two-phase process serves the purpose: We say that an edge of $G$ is *movable* if the embedding of $G$ can be changed by moving the edge into a face of $G$. For example, edges $(1, 2)$, $(2, 5)$, and $(5, 1)$ are the movable edges in the plane graph shown in Figure 2.2(a). Imagine that node $v$ is at the "bottom" of $G$. The first phase flips each movable incident edge of $v$ to the leftmost possible face. At the end of the first phase, the node $p$ is exactly the neighbor of $v$ on $\mathrm{cw}(G, r, v)$ that is closest to $r$. After determining $p$, the second phase flips each movable incident edge of $v$ that is to the left of edge $(p, v)$ to the rightmost possible face.

When recursively taking care of each 2-connected component $G_i$ of $G'$, the subroutine has to choose two distinct nodes $r_i$ and $v_i$ on the outer boundary of $G_i$ for the recursive subroutine call $\mathsf{block}(G_i, r_i, v_i)$. For any choice of $r_i$ and $v_i$, the subroutine call will return an orderly pair $(H_i, T_i)$ of $G_i$ with respect to $r_i$. However, to ensure that gluing all returned orderly pairs together yields an orderly pair of $G$, we have to be careful about the choice of each $v_i$.

The detailed description of $\mathsf{block}(G, r, v)$ is as follows.

**Subroutine block($G, r, v$).**

*Step* 1. If $G$ consists of a single edge $(r, v)$, then return $(G, G)$; otherwise, perform steps 2–7.

*Step* 2. Perform step 2.1 for each internal face $F$ incident to node $v$ in $G$ in clockwise order around $v$ starting from the one containing $(v, \mathrm{prev}(G, v))$.

- *Step* 2.1. For any node $x$ in $F$ such that $(v, x)$ is an edge of $G$ succeeding $F$ in clockwise order around $v$ starting from $(v, \mathrm{next}(G, v))$, update the planar embedding of $G$ by flipping $(v, x)$ into the interior of $F$.

  *Remark.* For instance, if $v$ and $F$ are as shown in Figure 2.4, then $(v, x_1)$ and $(v, x_2)$ will be flipped into the interior of $F$ by step 2.1.

*Step* 3. Let $p$ be the neighbor of $v$ in $G$ closest to $r$ in $\mathrm{cw}(G, r, v)$.

*Step* 4. Perform step 4.1 for each internal face $F$ of $G$ that succeeds $(v, p)$ in counterclockwise order around $v$ starting from the face containing $(v, p)$:

- *Step* 4.1. For any node $x$ in $F$ such that $(v, x)$ is an edge of $G$ succeeding $F$ in counterclockwise order around $v$ starting from $(v, \mathrm{next}(G, v))$, update the planar embedding of $G$ by flipping $(v, x)$ into the interior of $F$.

  *Remark.* For instance, if $v$ and $F$ are as shown in Figure 2.4, then $(v, x_3)$ and $(v, x_4)$ will be flipped into the interior of $F$ by step 4.1.

*Step* 5. Let $G'$ be the graph obtained by deleting all the incident edges of $v$ in $G$, except for $(v, p)$. Compute the 2-connected components of $G'$ by traversing the segment of the outer boundary of $G'$ from $\mathrm{prev}(G, v)$ to $\mathrm{next}(G, v)$ in counterclockwise order around the outer boundary of $G'$.

*Remark.* Since $G$ is 2-connected, we know that all 2-connected components of $G'$ are external to one another. Therefore, the above traversal of part of the outer boundary will suffice. Also, by the definitions of $G'$ and $p$, one of the 2-connected components of $G'$ consists of the single edge $(v, p)$.

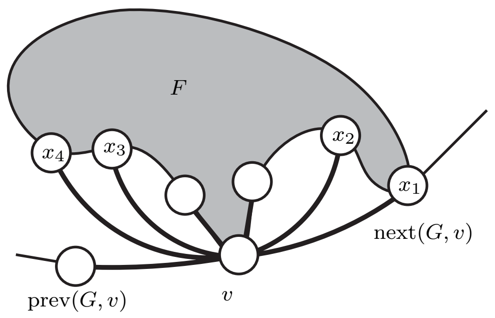


FIG. 2.4. *$F$ is an internal face of $G$ containing nodes $v$ and $x_i$, but not edge $(v, x_i)$ for each $i \in \{1, 2, 3, 4\}$.*

*Step* 6. Compute $(H_i, T_i) = \mathsf{block}(G_i, r_i, v_i)$ for each 2-connected component $G_i$ of $G'$, where $r_i$ is the node of $G_i$ closest to $r$ in $G'$, and $v_i$ is defined as follows:
*Case* 1. $G_i = (v, p)$. Let $v_i = v$.
*Case* 2. $G_i$ and $\text{prev}(G, v)$ are on the same side of $(v, p)$ in $G$. Let $S$ consist of the nodes in both $\text{ccw}(G_i, \text{next}(G_i, r_i), \text{prev}(G_i, r_i))$ and $\text{ccw}(G, r, v)$. If $S$ is empty, then let $v_i = \text{next}(G_i, r_i)$. Otherwise, let $v_i$ be the last node of $S$ in counterclockwise order around the outer boundary of $G_i$.
*Case* 3. $G_i$ and $\text{next}(G, v)$ are on the same side of $(v, p)$ in $G$. Let $S$ consist of the nodes in both $\text{ccw}(G_i, \text{next}(G_i, r_i), \text{prev}(G_i, r_i))$ and $\text{cw}(G, r, v)$. If $S$ is empty, then let $v_i = \text{prev}(G_i, r_i)$. Otherwise, let $v_i$ be the first node of $S$ in counterclockwise order around the outer boundary of $G_i$.

*Step* 7. Return $(H, T)$, where $H$ is obtained from $G$ by replacing each $G_i$ with $H_i$, and $T$ is the union of all $T_i$.

An illustration of $\mathsf{block}(G, r, v)$ is given in Figure 2.5. Let $G$ be the 2-connected plane graph shown in Figure 2.5(a). At the completion of step 4, the resulting embedding of $G$ and $p$ are as shown in Figure 2.5(b), where the gray ellipse with label $i$ is the $i$th 2-connected component $G_i$ of $G'$. Note that $(v, p)$ is also a 2-connected component of $G'$. One can verify that after step 6 we have that $r_1 = r$, $r_2 = r_6$, $r_8 = r_9$, $r_{11} = r_{12} = p$, and $v_{11} = v$. For the 2-connected components lying on the same side of $(v, p)$ with $\text{prev}(G, v)$, we have $v_1 = r_2$, $v_2 = r_3$, $v_3 = r_4$, $v_4 = \text{prev}(G, v)$, and that $v_i = \text{next}(G_i, r_i)$ holds for each $i \in \{5, 6, \ldots, 10\}$. For the 2-connected components lying on the same side of $(v, p)$ with $\text{next}(G, v)$, we have $v_{12} = r_{13}$, $v_{13} = r_{15}$, $v_{14} = \text{prev}(G_{14}, r_{14})$, and $v_{15} = \text{next}(G, v)$.

LEMMA 2.1. *If $r$ and $v$ are two distinct external nodes of a 2-connected plane graph $G$, then $\mathsf{block}(G, r, v)$ outputs an orderly pair of $G$ with respect to $r$.*

*Proof.* Let $(H, T)$ be the output of $\mathsf{block}(G, r, v)$. We prove the following properties of $(G, H, T, r, v)$ by induction on the number of edges in $G$:

1. Each external node of $G$ remains external in $H$. Moreover, $\text{boundary}(G, r)$ is a subsequence of $\text{boundary}(H, r)$.
2. (Property $\Pi$.) For each neighbor $x$ of $v$ in $H$ other than $p$, if $x$ and $\text{prev}(H, v)$ (respectively, $\text{next}(H, v)$) are on the same side of $(v, p)$ in $H$, then $(v, x)$ is on the first (respectively, last) internal face of $H$ containing $v$ and $x$ in counterclockwise order around $v$ starting from the one containing $(v, \text{next}(H, v))$.
3. $T$ rooted at $r$ is a spanning tree of $H$ such that exactly one of the following

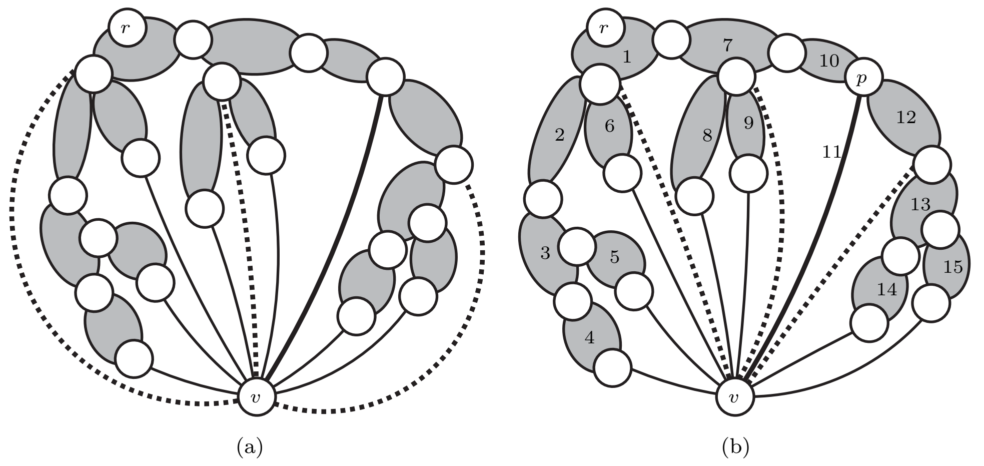


FIG. 2.5. (a) *A* 2*-connected plane graph $G$, where each gray ellipse is a* 2*-connected component of $G - \{v\}$.* (b) *The plane graph $G$ at the completion of performing steps* 1–4 *of* $\mathsf{block}(G, r, v)$*. Observe that all edges incident to $v$, especially those dashed* (*i.e., movable*) *edges, satisfy property* Π.

conditions holds for each node $u$ in $\text{ccw}(H, r, v)$ (respectively, $\text{cw}(H, r, v)$): (i) $u$ is a leaf of $T$; or (ii) $\text{next}(H, u)$ (respectively, $\text{prev}(H, u)$) is the lowest-indexed (respectively, highest-indexed) child of $u$ in $T$.

4. $H \sim G$.
5. $T$ rooted at $r$ is an orderly spanning tree of $H$.

Properties 4 and 5 suffice, but we need the other properties to enable the induction step. When $G$ consists of a single edge $(r, v)$, by step 1 we have $H = T = G$. It is not difficult to see the inductive basis of each property holds. Suppose that $G'$ consists of $k$ 2-connected components. By step 6, we have $r_i \neq v_i$ for each $i$. It follows from the inductive hypothesis that properties 1–5 of $(G_i, H_i, T_i, r_i, v_i)$ hold for each $i \in \{1, 2, \ldots, k\}$. The rest of the proof shows the induction step. For brevity, for each $j = 1, 2, \ldots, 5$, we abbreviate "property $j$ of $(G_i, H_i, T_i, r_i, v_i)$" to "property $j$ of $G_i$" and use "property $j$ (of $G$)" to stand for "property $j$ of $(G, H, T, r, v)$."

*Property* 1. Observe that throughout the execution of $\mathsf{block}(G, r, v)$, without accounting for its subsequent subroutine calls to $\mathsf{block}$, the embedding of $G$ changes only by flipping edges into the interior of internal faces of $G$ in steps 2 and 4. Thus, based on how $H$ is obtained from $G$ in step 7, it follows from property 1 of $G_i$ for each $i \in \{1, 2, \ldots, k\}$ that the property holds.

*Property* 2. Let property $2'$ stand for the property obtained from property 2 by replacing each $H$ with a $G$. From steps 2 and 4, one can verify that the plane graph $G$ at the completion of performing step 4 satisfies property $2'$. From property 2 and how $H$ is obtained from $G$ in step 7, we know that the relative order among the incident edges of $v$ and the faces containing $v$ remains the same in $G$ and $H$. Therefore the property follows from property $2'$.

*Property* 3. For each $i \in \{1, 2, \ldots, k\}$, property 3 of $G_i$ implies that $T_i$ is a spanning tree of $H_i$. Since $H_1, H_2, \ldots, H_k$ are edge disjoint, and each node of $H$ belongs to some $H_i$, we know that $T$, the union of all $T_i$, is a spanning tree of $H$. Since $v$ is a leaf of $T$, the required property holds for $v$. Let $x$ be an external node of $H$ other than $v$. If $(x, v)$ is not an external edge of $H$ belonging to $H - T$, then the required property for $x$ follows from the property of $x$ guaranteed by property 3 of

$G_i$ for each index $i$ with $x \in H_i$. Otherwise, by property 2, $x$ is either $\text{prev}(H, v)$ or $\text{next}(H, v)$. Let $H_j$ be the 2-connected component of $H'$ containing $x$. We have that $v_j = x$. By property 3 of $G_j$, $x$ is a leaf of $T_j$. Since $(x, v) \notin T$, $x$ is also a leaf of $T$ and property 3 holds for $x$.

*Property* 4. Observe that steps 2 and 4 flip an edge $(v, x)$ into the interior of $F$ only if $F$ contains both $v$ and $x$. Therefore, the resulting embedding of $G$ at the completion of step 4 is still planar. According to how $H$ is obtained from $G$ in step 7, the property follows from property 1 of $G$ and properties 4 of $G_i$ for all indices $i \in \{1, 2, \ldots, k\}$.

*Property* 5. Each neighbor of $r$ in $H$ is a child of $r$ in $T$; hence $r$ is orderly in $H$ with respect to $T$. The rest of the proof shows that each node $x$ other than $r$ is orderly in $H$ with respect to $T$. Let $H'$ be the graph obtained from $H$ by deleting each incident edge of $v$ in $H - T$. Observe that $H' \sim G'$ and that each $H_i$ is a 2-connected component of $H'$. Let $I_x$ consist of the indices $i$ with $x \in H_i$. As $x \neq r$, one can verify that there is an index $j$ in $I_x$ such that $x \neq r_j$ and $x = r_i$ for each index $i \in I_x - \{j\}$.

We first show that if $(v, x)$ is an edge of $H - H'$, then $(v, x)$ is unrelated with respect to $T$. If the index of $x$ is higher than that of $v$, then by the fact that $v$ is a leaf in $T$, we know that $(v, x)$ is unrelated. As for the case with the index of $x$ lower than that of $v$, let us assume for a contradiction that $x$ is an ancestor of $v$ in $T$. Since $(v, x) \in H - T$ and $p$ is the parent of $v$ in $T$, we know that $x$ is also an ancestor of $p$ in $T$. Let $P$ be the path of $T$ between $r$ and $p$. Thus, $x \in P$. Let $y$ be the node of $H_j$ closest to $p$ in $P$. It is not difficult to see that $y \in \text{cw}(H, r, p)$ and $y \in \text{cw}(H_j, r_j, x)$. Since $(v, p) \in T$, $(v, x) \in H - T$, and $y \in \text{cw}(H, r, p)$, we know $y \neq x$. Otherwise, $x$ would have been a neighbor of $v$ in $H$ closer to $r$ than $p$ in $\text{cw}(H, r, v)$, thereby contradicting the choice of $p$ in step 3. As $y \neq x$, $x$ is not a leaf of $T_j$. By step 6(2), $x \in \text{cw}(H_j, r_j, v_j)$. Let $z = \text{prev}(H_j, x)$. By property 3 of $G_j$, node $z$ has to be the largest-indexed child of $x$ in $T_j$. Since $x \neq r_j$ and $y \in \text{cw}(H_j, r_j, x)$, we know that $y$ and $z$ are on different sides of the path of $T_j$ between $r_j$ and $x$ in $H_j$, thereby contradicting the fact that $z$ is the highest-indexed child of $x$ in $T_j$.

We then show that $x$ is orderly in $H'$ with respect to $T$. If $|I_x| = 1$, then the orderly pattern of $x$ in $H'$ with respect to $T$ follows immediately from that in $H_j$ with respect to $T_j$, which is ensured by property 5 of $G_j$. When $|I_x| \geq 2$, by properties 5 of $G_i$ for all $i \in I_x - \{j\}$, each neighbor of $x$ in $\bigcup_{i \in I_x - \{j\}} H_i$ is a child of $x$ in $T$. It follows from property 3 of $G_j$ that all children of $x$ in $T$ are consecutive in $H'$ around $x$. Since $x$ is orderly in $H_j$ with respect to $T_j$, one can see that $x$ is orderly in $H'$ with respect to $T$.

Since $v$ is a leaf of $T$, we know that $v$ is orderly in $H$ with respect to $T$. It remains to show that each neighbor $x$ of $v$ in $H - H'$ is orderly in $H$ with respect to $T$. Let $z_1$ (respectively, $z_2$) be the neighbor of $x$ that precedes (respectively, succeeds) $v$ in counterclockwise order around $x$. It suffices to show that if the index of $x$ is lower (respectively, higher) than that of $v$, then $z_2$ (respectively, $z_1$) belongs to $P(x)$ or $D(x)$ (respectively, $P(x)$ or $U_<(x)$) of $H'$ with respect to $T$ as follows: If the index of $x$ is lower than that of $v$, then $z_2 = \text{next}(H_j, x)$ by property 2. By step 6, one can verify that $x$ belongs to $\text{cw}(H_j, r_j, v_j)$. By property 3, we have that $z_2$ belongs to either $P(x)$ or $D(x)$ of $H'$ with respect to $T$. If the index of $x$ is higher than that of $v$, then we know $z_1 = \text{prev}(H_j, x)$ from property 2. By step 6, one can verify that $x$ belongs to $\text{ccw}(H_j, r_j, v_j)$. From property 3, we have that $z_1$ belongs to either $P(x)$ or $U_<(x)$ of $H'$ with respect to $T$. ◻

LEMMA 2.2. *If $r$ and $v$ are two distinct external nodes of an $n$-node 2-connected* *plane graph $G$, then* $\mathsf{block}(G, r, v)$ *runs in $O(n)$ time.*

*Proof.* The execution of $\mathsf{block}(G, r, v)$ consists of a sequence of subroutine calls to $\mathsf{block}$. One can see that each node of $G$ can be the parameter $v$ for no more than two subroutine calls to $\mathsf{block}$—one with $G \neq (r, v)$ and the other with $G = (r, v)$. If $G = (r, v)$, then the subroutine call $\mathsf{block}(G, r, v)$ runs in $O(1)$ time. Let $\ell$ be the number of subroutine calls to $\mathsf{block}(G, r, v)$ with $G \neq (r, v)$. For each $j \in \{1, 2, \ldots, \ell\}$, let $\mathsf{block}(G^j, r^j, v^j)$ be the $j$th subroutine call to $\mathsf{block}$ with $G^j \neq (r^j, v^j)$ throughout the execution of $\mathsf{block}(G, r, v)$, where $G^1 = G$, $r^1 = r$ and $v^1 = v$. Clearly, $v^j \neq v^{j'}$ holds for any two distinct indices $j$ and $j'$, thereby implying that $\ell \leq n$. Let $E_j$ consist of the edges of $G$ belonging to the boundaries of the internal faces of $G^j$ that contain $v^j$. Let $t_j$ be the time required by $\mathsf{block}(G^j, r^j, v^j)$, without accounting for that required by its subsequent subroutine calls to $\mathsf{block}$. Observe that $t_j = O(|E_j|)$ holds for each $j$. It is not difficult to implement the algorithm $\mathsf{block}$ such that the running time of $\mathsf{block}(G, r, v)$ is dominated by $\sum_{j=1}^{\ell} t_j = \sum_{j=1}^{\ell} O(|E_j|)$. Since $G$ has $O(n)$ edges, it suffices to show as follows that any edge $(x, y)$ of $G$ belongs to no more than two of the sets $E_1, E_2, \ldots, E_\ell$: Let $j_1$ be the smallest index $j$ with $(x, y) \in E_j$. If $v^{j_1} \in \{x, y\}$, then $j_1$ is also the largest index $j$ with $(x, y) \in E_j$. It remains to consider the case $v^{j_1} \notin \{x, y\}$. Let $j_2$ be the smallest index $j$ with $j > j_1$ and $(x, y) \in E_j$. From the definition of $\mathsf{block}$, edge $(x, y)$ has to be on the outer boundary of $G^{j_2}$, implying $v^{j_2} \in \{x, y\}$. Therefore, $j_2$ is the largest index $j$ with $(x, y) \in E_j$. □

Finally, we have the next theorem.

THEOREM 2.3. *It takes $O(n)$ time to compute an orderly pair for an $n$-node connected planar graph.*

## 3. Realizers for plane triangulations.

This section provides a new linear-time algorithm for computing a realizer for any $n$-node plane triangulation $\Delta$. As defined by Schnyder [39, 38], $(T_1, T_2, T_n)$ is a *realizer* of $\Delta$ if

- the internal edges of $\Delta$ are partitioned into three edge-disjoint trees $T_1$, $T_2$, and $T_n$, each rooted at a distinct external node of $\Delta$; and
- the neighbors of each internal node $v$ of $\Delta$ form six blocks $U_1$, $D_n$, $U_2$, $D_1$, $U_n$, and $D_2$ in counterclockwise order around $v$, where for each $j \in \{1, 2, n\}$, $U_j$ (respectively, $D_j$) consists of the parent (respectively, children) of $v$ in $T_j$.

A realizer of the plane triangulation in Figure 1.1(a) is shown in Figure 1.1(c).

LEMMA 3.1. *Given an orderly spanning tree of $\Delta$, a realizer of $\Delta$ is computable in $O(n)$ time.*

*Proof.* Let $T$ be the given orderly spanning tree of $\Delta$ rooted at $v_1$. Let $v_1, \ldots, v_n$ be the counterclockwise preordering of $T$, where $v_1$, $v_2$, and $v_n$ are the external nodes of $\Delta$ in counterclockwise order. Note that $(v_1, v_2)$ and $(v_1, v_n)$ must be in $T$. Since $\Delta$ is a plane triangulation and the edges of $\Delta - T$ are unrelated with respect to $T$, both $U_<(v_i)$ and $U_>(v_i)$ are nonempty for each $3 \leq i \leq n-1$. (To see this, one can verify that if $U_<(v_i)$ or $U_>(v_i)$ were empty, then the edge between $v_i$ and the parent of $v_i$ in $T$ would belong to a face of $\Delta$ consisting of at least four edges, contradicting the assumption that $\Delta$ is a plane triangulation.) Let $p_i$ (respectively, $q_i$) be the index of the last (respectively, first) node in $U_<(v_i)$ (respectively, $U_>(v_i)$) in counterclockwise order around $v_i$. Let $T_1$ be obtained from $T$ by deleting $(v_1, v_2)$ and $(v_1, v_n)$. Let $T_2 = \{(v_i, v_{p_i}) \mid 3 \leq i \leq n-1\}$ and $T_n = \{(v_i, v_{q_i}) \mid 3 \leq i \leq n-1\}$. An example is shown in Figure 1.1(c). Observe that $p_i < i < q_i$ holds for each $3 \leq i \leq n-1$, implying that both $T_2$ and $T_n$ are acyclic. It can be proved as follows that exactly one of the equalities $i = p_j$ and $j = q_i$ holds for each edge $(v_i, v_j) \in \Delta - T$ with $i < j$.

> Since each face of $\Delta$ has size three, there is a node $v_k$ that is (i) the neighbor of $v_i$ immediately succeeding $v_j$ in clockwise order around $v_i$ and (ii) the neighbor of $v_j$ immediately preceding $v_i$ in clockwise order around $v_j$. Clearly, $i < j < k$ implies $i = p_j$ and $j \neq q_i$; and $k < i < j$ implies $j = q_i$ and $i \neq p_i$. As for the remaining case that $i < j < k$, one can verify that $v_i$ has to be the parent of $v_k$ in $T$, thereby implying $j = q_i$ and $i \neq p_j$.

It follows that each internal edge of $\Delta$ belongs to exactly one of $T_1$, $T_2$, and $T_n$. By the definitions of $p_i$ and $q_i$, one can verify that the neighbors of each internal node $v_i$ of $\Delta$ indeed form the required pattern for $(T_1, T_2, T_n)$ as a realizer of $\Delta$. Since it takes $O(n)$ time to determine all $p_i$ and $q_i$, the lemma is proved. □

THEOREM 3.2 (see also [39, 38]). *A realizer of any plane triangulation is derivable in linear time.*

*Proof.* The proof is straightforward by Theorem 2.3 and Lemma 3.1. □

**4. 2-visibility drawings for plane graphs.** This section shows how to obtain in $O(n)$ time an $(n-1) \times \lfloor \frac{2n+1}{3} \rfloor$ 2-visibility drawing for any $n$-node plane graph $G$. For calculating the area of a 2-visibility drawing, we follow the convention of [16], stating that the corner coordinates of each rectangle are integers, and that each rectangle is no smaller than $1 \times 1$. For example, the area of the 2-visibility drawing shown in Figure 1.1(b) is $9 \times 8$. Let $\Delta$ be a plane triangulation obtained by triangulating $G$. Since any 2-visibility drawing of $\Delta$ is also a 2-visibility drawing of $G$, the rest of the section assumes that the input is the plane triangulation $\Delta$.

Let $T$ be an orderly spanning tree of $\Delta$. Let $v_1, v_2, \ldots, v_n$ be the counterclockwise preordering of the nodes in $T$. Our algorithm $\mathsf{draw}(\Delta, T)$ consists of $n$ iterations, where the $i$th iteration performs the following steps:

*Step* 1. If $i \neq 1$ and $v_i$ is not the first child of its parent in $T$, then lengthen each ancestor of $v_i$ in $T$ to the right by one unit.

*Step* 2. Draw $v_i$ as a unit square beneath the parent of $v_i$ in $T$ such that $v_i$ and all ancestors of $v_i$ in $T$ align along the right boundary. Now, $v_i$ is vertically visible to its parent in $T$.

*Step* 3. Lengthen downward $v_i$ and each neighbor $v_j$ of $v_i$ in $\Delta$ with $j < i$, if necessary, so that $v_i$ and $v_j$ are horizontally visible to each other.

If $\Delta$ and $T$ are as shown in Figure 4.1(a), then the intermediate (respectively, resulting) drawing obtained by $\mathsf{draw}(\Delta, T)$ is shown in Figure 4.1 (respectively, Figure 4.1(b)).

LEMMA 4.1. *The algorithm* $\mathsf{draw}(\Delta, T)$ *obtains an* $h \times w$ *2-visibility drawing of* $\Delta$ *with* $h \leq n - 1$ *and such that* $w$ *equals the number of leaves in* $T$.

*Proof.* Since $T$ is a spanning tree of $\Delta$, $\mathsf{draw}(\Delta, T)$ is well defined. Also, the output of $\mathsf{draw}(\Delta, T)$ is indeed a 2-visibility drawing of $\Delta$ with width equal to the number of leaves in $T$. The rest of the proof shows that the height of the output of $\mathsf{draw}(\Delta, T)$ is at most $n - 1$. For any two distinct edges $e$ and $e'$ in $\Delta - T$, we say that $e$ *encloses* $e'$ if $e'$ is in the interior of the cycle consisting of $e$ and the path of $T$ between the endpoints of $e$. By the planarity of $\Delta$, the above "enclosing" relation defines a nesting structure among the edges of $\Delta - T$. Let $\ell(e)$ denote the "level" of edge $e$ in the nesting structure:

> If $e$ does not enclose any other edge in $\Delta - T$, then let $\ell(e) = 1$; otherwise, let $\ell(e)$ be one plus the maximum of $\ell(e')$ over all the edges $e'$ in $\Delta - T$ that are enclosed by $e$.

If we are to draw edge $e = (u, v)$ as a horizontal line segment connecting the rectangles

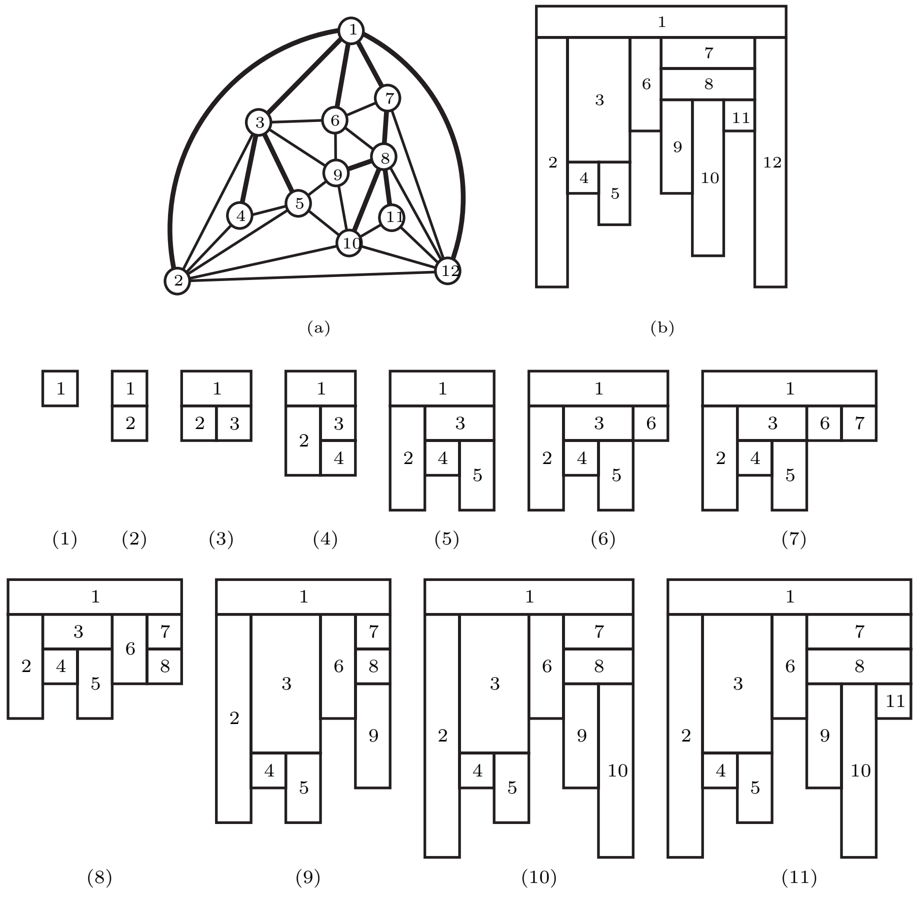


FIG. 4.1. *An illustration of the intermediate steps of* draw$(\Delta, T)$, *where* $\Delta$ *and* $T$ *are as shown in* (a) *and the final drawing is as shown in* (b).

representing nodes $u$ and $v$ without intersecting other rectangles or line segments, then $1 + \ell(e)$ is the minimum possible (vertical) distance between the line segment representing $e$ and the rectangle representing the lowest common ancestor of $u$ and $v$ in $T$. Let edge $\hat{e}$ be $(v_2, v_n)$, which encloses all the other edges in $\Delta - T$. The height of the output of draw$(\Delta, T)$ is exactly $1 + \ell(\hat{e})$. It remains to show $\ell(\hat{e}) \leq n - 2$ as follows: Assume for the sake of contradiction that $e_1, e_2, \ldots, e_{n-1}$ is a sequence of edges in $\Delta - T$ such that $e_i$ encloses $e_1, e_2, \ldots, e_{i-1}$ for each $i \in \{2, 3, \ldots, n-1\}$. For each $i \in \{1, 2, \ldots, n-1\}$, let $X_i$ consist of the endpoints of $e_i, e_{i+1}, \ldots, e_{n-1}$. For each $i \in \{1, 2, \ldots, n-2\}$, there must be an endpoint of $e_i$ that is not in $X_{i+1}$. (To see this, assume for the sake of contradiction that $j$ and $k$ with $i < j < k \leq n-1$ are two indices such that $e_j$ and $e_k$ are incident to the endpoints $u$ and $v$ of $e_i$, respectively. Let $C$ be the cycle consisting of $e_j$ and the path of $T$ that connects the endpoints of $e_j$. Since $e_i$ is in the interior of $C$ and $e_k$ is outside of $C$, the assumption that $e_k$ is incident to $v$ violates the orderly property of $v$.) Therefore, $X_1$ contains at least $n$ distinct nodes. Since $T$ is an orderly spanning tree of $\Delta$, $v_1$ is not incident to any edges of $\Delta - T$. Therefore, $v_1 \notin X_1$, which contradicts that $\Delta$ has $n$ nodes. ◻

LEMMA 4.2. *It takes $O(n)$ time to compute an orderly spanning tree of $\Delta$ with*

$\lfloor \frac{2n+1}{3} \rfloor$ *or fewer leaves.*

*Proof.* Let $v_1$, $v_2$, and $v_n$ be the external nodes of $\Delta$ in counterclockwise order around the outer boundary of $\Delta$. By Theorem 3.2, a realizer $(T'_1, T'_2, T'_n)$ of $\Delta$, where each $T'_i$ is rooted at $v_i$, can be obtained in $O(n)$ time. Let $I = \{1, 2, n\}$. For each $i \in I$, let $T_i = T'_i \cup \{(v_i, v_{i_1}), (v_i, v_{i_2})\}$, where $\{i_1, i_2\} = I - \{i\}$. Observe that $T_1$, $T_2$, and $T_n$ are three spanning trees of $\Delta$ with $T_1 \cup T_2 \cup T_n = \Delta$. We first show that each $T_i$ is an orderly spanning tree of $\Delta$. Since the relation between $T_1$, $T_2$, and $T_n$ is rotationally symmetric, it suffices to verify that each node is orderly with respect to $T_1$. Let $v_1, v_2, \ldots, v_n$ be the counterclockwise preordering of $T_1$. For each $i \in I$ and $j \in \{1, 2, \ldots, n\}$, let $Q_{i,j}$ be the path of $T_i$ between $v_i$ and $v_j$. Note that $Q_{1,j}$, $Q_{2,j}$, and $Q_{n,j}$ are three edge-disjoint paths of $\Delta$ that intersect only at $v_j$. If $v_j$ is not a leaf of $T_1$, then the children of $v_j$ in $T_1$ are consecutive in $\Delta$ in counterclockwise order around $v_j$. Therefore, to ensure that each node is orderly with respect to $T_1$, it suffices to prove that each edge of $\Delta - T_1$ is unrelated with respect to $T_1$: If $v_{j'}$ were an ancestor of $v_j$, that is, also a neighbor of $v_j$ in $\Delta - T_1$, then $v_j$ and $v_{j'}$ would be on different sides of $Q_{2,j''} \cup Q_{n,j''}$ in $\Delta$, where $v_{j''}$ is the parent of $v_j$ in $T_1$, thereby contradicting the planarity of $\Delta$.

It remains to show that $T_1$, $T_2$, or $T_n$ has at most $\frac{2n+1}{3}$ leaves. For each $i \in I$, let leaf$(T'_i)$ consist of the leaves of $T'_i$. Since the number of leaves in $T_i$ is precisely $2 + |\text{leaf}(T'_i)|$, it suffices to show that $\sum_{i \in I} |\text{leaf}(T'_i)| \leq 2n - 5$. Let $v$ be a node in leaf$(T'_i)$. Observe that $v$ is internal in $\Delta$. For each $i \in I$, let $p_i(v)$ denote the parent of $v$ in $T_i$. Let $i_1$ and $i_2$ be the indices in $I - \{i\}$. Since $(T'_1, T'_2, T'_n)$ is a realizer of $\Delta$, there is a unique internal face $F_i(v)$ of $\Delta$ containing $v$, $p_{i_1}(v)$, and $p_{i_2}(v)$. We have that $p_{i_1}(v) \notin \text{leaf}(T'_{i_1})$ and $p_{i_2}(v) \notin \text{leaf}(T'_{i_2})$. It follows that $F_i(v) \neq F_{i_1}(u_1)$ for any node $u_1$ in leaf$(T'_{i_1})$ and that $F_i(v) \neq F_{i_2}(u_2)$ for any node $u_2$ in leaf$(T'_{i_2})$. Therefore, $\sum_{i \in I} |\text{leaf}(T'_i)|$ is no more than the number of internal faces of $\Delta$, which is precisely $2n - 5$ by Euler's formula. □

THEOREM 4.3. *An* $(n-1) \times \lfloor \frac{2n+1}{3} \rfloor$ *2-visibility drawing of any n-node planar graph is computable in* $O(n)$ *time.*

*Proof.* A naive implementation of algorithm draw runs in $O(n^2)$ time, since one stretching of a node's rectangle affects the horizontal spans of all its descendants. However, it is not difficult to implement the algorithm to run in $O(n)$ time. The width of the rectangle representing a node $v$ is exactly the number of leaves in the subtree of $T$ rooted at $v$. Therefore, one can easily calculate the $x$-coordinates of all rectangles in linear time. Also, the $y$-coordinates of all rectangles can be computed in linear time via the values $\ell(e)$ for all edges $e$ in $G - T$. (See, e.g., Figure 3.2 of [32] for related implementation techniques.) Therefore, the theorem follows from Lemmas 4.1 and 4.2. □

## 5. Convenient encodings for planar graphs.

This section gives the best known convenient encoding for an $n$-node $m$-edge planar graph as an application of our orderly pair algorithm. We need some notation to describe the data structures required by our convenient encodings. Let $|S|$ denote the length of a string $S$, i.e., the number of symbols in $S$. Unless clearly stated otherwise, all strings in this section have length $O(m + n)$. A string $S$ consisting of $t$ distinct symbols can be encoded in $|S| \lceil \log_2 t \rceil$ bits. For example, if $S$ consists of parentheses and brackets, including open and closed ones, then $S$ can be encoded in $2|S|$ bits. $S$ is *binary* if it consists of two distinct symbols. For each $1 \leq i \leq j \leq |S|$, let $S[i, j]$ be the length-$(j - i + 1)$ substring of $S$ from the $i$th position to the $j$th position. If $i > j$, then let $S[i, j]$ be the empty string. Define $S[i] = S[i, i]$. $S[k]$ is *enclosed* by $S[i]$ and $S[j]$ in $S$ if $i < k < j$.

Let select$(S, i, \Box)$ be the position of the $i$th $\Box$ in $S$. Let rank$(S, k, \Box)$ be the number of $\Box$'s before or at the $k$th position of $S$. If $k = \text{select}(S, i, \Box)$, then $i = \text{rank}(S, k, \Box)$.

An *auxiliary string* $\chi$ of $S$ is a binary string with $|\chi| = o(|S|)$ which is obtainable from $S$ in $O(|S|)$ time.

FACT 5.1 (see [1, 15]). *For any strings $S_1, S_2, \ldots, S_k$ with $k = O(1)$, there is an auxiliary string $\chi_0$ such that, given the concatenation of $\chi_0, S_1, S_2, \ldots, S_k$ as input, the index of the first symbol of any given $S_i$ in the concatenation is computable in $O(1)$ time.*

Let $S_1 \circ S_2 \circ \cdots \circ S_k$ denote the concatenation of $\chi_0, S_1, S_2, \ldots, S_k$ as in Fact 5.1.

Suppose that $S$ is a string of multiple types of parentheses. Let reverse$(S)$ be the string $R$ such that $R[i]$ is the opposite parenthesis of the same type as $S[|S|+1-i]$. For example, reverse("`)()]])[`") = "`](([()(`." For an open parenthesis $S[i]$ and a closed one $S[j]$ of the same type with $i < j$, the two *match* in $S$ if every parenthesis of the same type that is enclosed by them matches one enclosed by them. $S$ is *balanced in type $k$* if every parenthesis of type $k$ in $S$ belongs to a matching parenthesis pair. $S$ is *balanced* if $S$ is empty or is balanced in all types of parentheses. Here are some queries defined for a balanced string $S$. Let match$(S, i)$ be the position of the parenthesis in $S$ that matches $S[i]$. Let enclose$_k(S, i_1, i_2)$ be the position pair $(j_1, j_2)$ of the closest matching parenthesis pair of the $k$th type that encloses $S[i_1]$ and $S[i_2]$.

FACT 5.2 (see [35, 9]). *For any balanced string $S$ of $O(1)$ types of parentheses, there is an auxiliary string $\chi_1(S)$ such that each of* rank$S, i, \Box$, *select$(S, i, \Box)$, match$(S, i)$, and enclose$_k(S, i, j)$ can be determined from $S \circ \chi_1(S)$ in $O(1)$ time.*

For a string $S$ of parentheses that may be unbalanced, we define wrapped$(S, i)$ as follows. For the case that $S[i]$ is an open parenthesis of type $k$, let $S'$ be a string obtained from $S$ by appending some closed parentheses of type $k$, if necessary, such that $S'[i]$ is matched in $S'$. Define wrapped$(S, i)$ to be the number of indices $j$ satisfying $i < j \le |S|$, $\text{enclose}_k(S', j, \text{match}(S', j)) = (i, \text{match}(S', i))$, and $S[j]$ is of type $k$. For the case that $S[i]$ is closed, let $\text{wrapped}(S, i) = \text{wrapped}(\text{reverse}(S), |S| + 1 - i)$. Therefore, if

(5.1) $S = \texttt{(()[[[[[(](])[(]])[[)[[(])[[[(](](]]])[(]]])[[(])[)[)[(]]]]]))},$

```
122.....3.4.4.5..5..3..6.6...7.8.9...9.A...A..B.B.8.7.C.....C1
```

then we have wrapped$(S, 1) = 10$ and wrapped$(S, 6) = 4$. If $S$ is balanced, then wrapped$(S, i)$ is an even number for each $i$, i.e., twice the number of parenthesis pairs that are enclosed by the parenthesis pair in question. The next lemma extends the set of queries supported in Fact 5.2.

LEMMA 5.3. *For any balanced string $S$ of $O(1)$ types of parentheses, there is an auxiliary string $\chi_2(S)$ such that wrapped$(S, i)$ can be computed from $S \circ \chi_2(S)$ in $O(1)$ time.*

*Proof.* Define $\text{width}(i, j) = |i - j| + 1$. We say that parentheses of the same type with property $\pi$ are *$d$-disjoint* if

- $\text{width}(i, \text{match}(S, i)) > d$ holds for any parenthesis $S[i]$ with property $\pi$; and
- any two property-$\pi$ parentheses $S[i]$ and $S[j]$ with $S[i] = S[j]$ satisfy at least one of $\text{width}(i, j) > d$ and $\text{width}(\text{match}(S, i), \text{match}(S, j)) > d$.

Intuitively, $d$-disjoint parentheses have to be sparse: Suppose that parentheses with property $\pi$ are $d$-disjoint. Then, for any property-$\pi$ open parenthesis $S[i]$, either $S[i]$ is the only property-$\pi$ open parenthesis in $S[i - d + 1, i]$ or $S[\text{match}(S, i)]$ is the only property-$\pi$ closed parenthesis in $S[\text{match}(S, i), \text{match}(S, i) + d - 1]$. As a result, suppose that we partition $S$ into segments of length $d$. For each segment, let us mark (a) the property-$\pi$ open parenthesis with the smallest index in the segment and

(b) the property-$\pi$ closed parenthesis with the largest index in the segment. Then, for each parenthesis $S[i]$ with property $\pi$, at least one of $S[i]$ and $S[\text{match}(S,i)]$ is marked. Therefore, there are only $O(s/d)$ parentheses with property $\pi$.

Let $s = |S|$. For a carefully chosen number $\ell = \Theta(\log s)$, we say that

- parenthesis $S[i]$ is *narrow* if $\text{width}(i, \text{match}(S,i)) \leq \ell$;
- parenthesis $S[i]$ is *wide* if $\text{wrapped}(S,i) > 2\ell^2$; and
- parenthesis $S[i]$ is *medium* if it is neither narrow nor wide.

We apply the commonly used preprocessing technique (see, e.g., [10, 35]) in the unit-cost RAM model which allows the query $\text{wrapped}(S,i)$ for any narrow $S[i]$ to be answered in $O(1)$ time from the linear-time precomputable $o(s)$-bit table (i.e., the table $M_1 \circ M_2$ to be described later). It is not difficult to see that wide parentheses are $\ell^2$-disjoint. Therefore, we can afford to encode $(i, \text{wrapped}(S,i))$ for all wide parentheses $S[i]$ using $o(s)$ bits. Although medium parentheses are not necessarily $\ell$-disjoint, we identify *special* parentheses, which are medium parentheses that have to be $\ell$-disjoint, and encode $(i, \text{wrapped}(S,i))$ for all special medium parentheses $S[i]$ using $o(s)$ bits. As for medium parentheses that are not special, we will show that two queries to the $o(s)$-bit precomputed table suffice. The details are as follows.

Let $t$ be the number of distinct types of parentheses in $S$. Let $b$ be the smallest integer with $2t \leq 2^b$. Each symbol of $S$ can be encoded in $b$ bits. As $t = O(1)$, we have $b = O(1)$. Let $\ell = \left\lfloor \frac{1}{2} \log_{2^b} s \right\rfloor$. Any substring $S[i,j]$ with $j \leq i+\ell-1$ has $O(\sqrt{s})$ possible distinct values. Define tables $M_1$ and $M_2$ for $S$ by letting $M_1[S[i,i+\ell-1]] = \text{wrapped}(S[i,i+\ell-1],1)$ and $M_2[S[i,j]] = \text{wrapped}(\text{reverse}(S[i,j]),1)$ for any $i,j$ with $1 \leq i \leq j \leq i+\ell-1$. One can easily come up with an $o(s)$-bit string $\chi_2'$ from which each entry of $M_1$ and $M_2$ can be obtained in $O(1)$ time.

For each $k \in \{1,2,\ldots,t\}$, define tables $M_3^k$ and $M_4^k$ as follows. For each $i = 1,2,\ldots,\left\lceil \frac{s}{\ell^2} \right\rceil$,

- let $M_3^k[i] = (j, \text{wrapped}(S,j))$, where index $j$ is the smallest index, if any, with $(i-1)\ell^2 < j \leq i\ell^2$ such that $S[j]$ is a wide open parenthesis of type $k$; and
- let $M_4^k[i] = (j, \text{wrapped}(S,j))$, where index $j$ is the largest index, if any, with $(i-1)\ell^2 < j \leq i\ell^2$ such that $S[j]$ is a wide close parenthesis of type $k$.

Since each entry of $M_3^k$ and $M_4^k$ can be encoded in $O(\log s)$ bits, one can easily obtain an $o(s)$-bit string $\chi_2''$ from which each entry of $M_3^k$ and $M_4^k$ can be determined in $O(1)$ time.

A medium open parenthesis $S[i]$ is *special* if at least one of the inequalities $\text{width}(i,j) > \ell$ and $\text{width}(\text{match}(S,i), \text{match}(S,j)) > \ell$ holds for each index $j$ with $i < j < \text{match}(S,i)$ and $S[j] = S[i]$. A closed parenthesis $S[i]$ is *special* if $S[\text{match}(S,i)]$ is special. One can verify that special parentheses are $\ell$-disjoint. For each $k \in \{1,2,\ldots,t\}$, define tables $M_5^k$ and $M_6^k$ as follows. For each $i \in \{1,2,\ldots,\left\lceil \frac{s}{\ell} \right\rceil\}$,

- let $M_5^k[i] = (j, \text{wrapped}(S,j))$, where $j$ is the smallest index, if any, with $(i-1)\ell < j \leq i\ell$ such that $S[j]$ is a special open parenthesis of type $k$; and
- let $M_6^k[i] = (j, \text{wrapped}(S,j))$, where $j$ is the largest index, if any, with $(i-1)\ell < j \leq i\ell$ such that $S[j]$ is a special close parenthesis of type $k$.

Observe that $M_5^k[i] = (j,c)$ or $M_6^k[i] = (j,c)$ implies $0 \leq i\ell - j \leq \ell$ and $0 \leq c \leq 2\ell^2$. Therefore, each entry of $M_5^k$ and $M_6^k$ can be encoded in $O(\log \ell) = O(\log\log s)$ bits. As a result, one can easily come up with an $o(s)$-bit string $\chi_2'''$ from which each entry of $M_5^k$ and $M_6^k$ can be determined in $O(1)$ time. Let $\chi_2(S) = \chi_2' \circ \chi_2'' \circ \chi_2'''$. The $o(s)$-bit string $\chi_2(S)$ can be derived from $S$ in $O(s)$ time.

It remains to show that $\text{wrapped}(S,i)$ can be determined from $S$ and $\chi_2(S)$ by

```
function wrapped$(S, i)$ {
    Step 1. let $k$, with $1 \leq k \leq t$, be the type of $S[i]$;
    Step 2. let $i_1 = \min\{i, \text{match}(S, i)\}$;
    Step 3. let $i_2 = \text{match}(S, i_1)$;
    Step 4. let $(j, c) = M_3^k\left[\lceil i_1/\ell^2 \rceil\right]$; if $j = i_1$, then return $c$;
    Step 5. let $(j, c) = M_4^k\left[\lceil i_2/\ell^2 \rceil\right]$; if $j = i_2$, then return $c$;
    Step 6. let $(j, c) = M_5^k[\lceil i_1/\ell \rceil]$; if $j = i_1$, then return $c$;
    Step 7. let $(j, c) = M_6^k[\lceil i_2/\ell \rceil]$; if $j = i_2$, then return $c$;
    Step 8. if width$(i_1, i_2) \leq \ell$, then return $M_1[S[i_1, i_2]]$;
    Step 9. if width$(i_1, i_2) \leq 2\ell$, then return $M_1[S[i_1, i_1 + \ell - 1]] + M_2[S[i_1 + \ell, i_2]]$;
    Step 10. return $M_1[S[i_1, i_1 + \ell - 1]] + M_2[S[i_2 - \ell + 1, i_2]]$;
}
```

FIG. 5.1. *An $O(1)$-time algorithm that computes* wrapped(S,i).

the algorithm shown in Figure 5.1, which clearly runs in $O(1)$ time. If a value $c$ is returned from steps 4–9, we have $c = \text{wrapped}(S, i)$. The rest of the proof assumes that step 10 is executed. Since wide parentheses are $\ell^2$-disjoint and special parentheses are $\ell$-disjoint, parentheses $S[i_1]$ and $S[i_2]$ at step 10 satisfy $\text{width}(i_1, i_2) > 2\ell$ and form a matching pair of medium parentheses that are not special. By definition of special parentheses, there are indices $j_1$ and $j_2$ with $i_1 < j_1 < j_2 = \text{match}(S, j_1) < i_2$ and $S[i_1] = S[j_1]$ (thus $S[j_2] = S[i_2]$) such that $\text{width}(i_1, j_1) \leq \ell$ and $\text{width}(j_2, i_2) \leq \ell$. Since $\text{width}(i_1, i_2) > 2\ell$, we have $\text{wrapped}(S, i_1) = M_1[S[i_1, i_1 + \ell - 1]] + M_2[S[i_2 - \ell + 1, i_2]]$. Therefore, step 10 correctly returns $\text{wrapped}(S, i)$. □

A folklore encoding [22, 35, 9] $S$ of an $n$-node simple rooted tree $T$ is a balanced string of $2n$ parentheses representing a counterclockwise depth-first traversal of $T$. Initially, an open (respectively, closed) parenthesis denotes a descending (respectively, ascending) edge traversal. Then, this string is enclosed by an additional matching parenthesis pair. For example, the string in (5.2) is the folklore encoding for the tree $T$ in Figure 1.1(a). Let $v_i$ be the $i$th node in the counterclockwise depth-first traversal. Let $(_i$ be the $i$th open parenthesis in $S$. Let $)_i$ be the closed parenthesis of $S$ that matches $(_i$ in $S$. Node $v_i$ corresponds to $(_i$ and $)_i$ in that $v_i$ is the parent of $v_j$ in $T$ if and only if $(_i$ and $)_i$ form the closest pair of matching parentheses that encloses $(_j$ and $)_j$. Also, the number of children of $v_i$ in $T$ is precisely $\text{wrapped}(S, \text{select}(S, i, ())/2$, which is also equal to $\text{wrapped}(S, \text{match}(S, \text{select}(S, i, ()))/2$.

Let $H$ be an $n$-node connected plane graph that may have multiple edges but no self-loops. Let $T$ be a spanning tree of $H$ rooted at $v_1$. Let $v_1 v_2 \cdots v_n$ be the counterclockwise preordering of $T$. Let degree($i$) be the number of edges incident to $v_i$ in $H$. Let children($i$) be the number of children of $v_i$ in $T$. Let above($i$) (respectively, below($i$)) be the number of edges $(v_i, v_j)$ of $H$ such that $v_j$ is the parent (respectively, a child) of $v_i$ in $T$. Let low($i$) (respectively, high($i$)) be the number of edges $(v_i, v_j)$ of $H$ such that $j < i$ (respectively, $j > i$) and $v_j$ is neither the parent nor a child of $v_i$ in $T$. Now, $\text{degree}(i) = \text{above}(i) + \text{below}(i) + \text{low}(i) + \text{high}(i)$. If $H$ has no multiple edges, then $\text{below}(i) = \text{children}(i)$. If $H$ and $T$ are as shown in Figure 1.1(a), for instance, then $\text{above}(3) = 1$, $\text{below}(3) = \text{children}(3) = 2$, $\text{low}(3) = 1$, $\text{high}(3) = 2$, and $\text{degree}(3) = 6$.

The *$T$-code* of $H$ is a triple $(S_1, S_2, S_3)$ of the binary strings $S_1, S_2, S_3$, where $\delta_{i \geq 2}$ equals 1 if $i \geq 2$ and 0 otherwise.

- $S_1$ is the folklore encoding of $T$.

- Let $p_i = \text{select}(S_1, i, \texttt{(})$ and $q_i = \text{match}(S_1, p_i)$. $S_2$ has exactly $2n$ copies of 1, in which low$(i)$ copies of 0 immediately succeed the $p_i$th 1, and high$(i)$ copies of 0 immediately succeed the $q_i$th 1.
- $S_3$ has exactly $n$ copies of 1, where above$(i)$ + below$(i)$ − children$(i)$ − $\delta_{i\geq 2}$ copies of 0 immediately succeed the $i$th 1.

For example, if $H$ and $T$ are as shown in Figure 1.1(a), then

$$S_1 = \texttt{(()(()())()((()()()))());} \tag{5.2}$$
$$S_2 = \texttt{11100000101010100100100101000101010001010001001010101010000011;}$$
$$S_3 = \texttt{111111111111.}$$

We have that

$$|S_1| = 2n;$$
$$|S_2| = 2n + \sum_{i=1}^{n} (\text{low}(i) + \text{high}(i));$$
$$|S_3| = 1 + \sum_{i=1}^{n} (\text{above}(i) + \text{below}(i) - \text{children}(i)).$$

Therefore, $|S_1| + |S_2| + |S_3| = 2m + 3n + 2$. Moreover, if $H$ has no multiple edges, then $|S_3| = n$, and thus $|S_1| + |S_2| = 2m + 2n + 2$.

The next theorem describes our convenient encoding. The techniques in the proof are mostly adapted from [9]. (Their encoding needs initial augmentation to the input graph to ensure that the resulting graph admits a canonical spanning tree. As a result, their encoding requires an additional number of bits to tell which edges are original.)

THEOREM 5.4. *Let $G$ be an input $n$-node $m$-edge planar graph having no self-loops. If $G$ has* (*respectively, has no*) *multiple edges, then $G$ has a convenient encoding, obtainable in $O(m+n)$ time, with $2m + 3n + o(m+n)$* (*respectively, $2m + 2n + o(n)$*) *bits.*

*Proof.* We focus on the case that $G$ is connected. As sketched at the end of the proof, it is not difficult to remove this restriction. By Theorem 2.3, an orderly pair $(H, T)$ of $G$ can be derived in $O(m + n)$ time. Let $(S_1, S_2, S_3)$ be the $T$-code of $H$. We prove that there exists an $o(m + n)$-bit string $\chi$, obtainable in $O(m + n)$ time, such that $S_1 \circ S_2 \circ S_3 \circ \chi$ is a convenient encoding of $G$. If $G$ has no multiple edges, then $S_3$ consists of $n$ copies of 1, and thus $S_1 \circ S_2 \circ \chi$ will suffice.

To support degree queries, let $p_i = \text{select}(S_1, i, \texttt{(})$ and $q_i = \text{match}(S_1, p_i)$. Since $S[p_i]$ and $S[q_i]$ are a matching parenthesis pair, we have that

$$\text{low}(i) = \text{select}(S_2, p_i + 1, 1) - \text{select}(S_2, p_i, 1) - 1,$$
$$\text{high}(i) = \text{select}(S_2, q_i + 1, 1) - \text{select}(S_2, q_i, 1) - 1.$$

Observe that children$(i)$ = wrapped$(S_1, p_i)/2$. From the definition of $S_3$, we know above$(i)$ + below$(i)$ − children$(i)$ = select$(S_3, i + 1, 1)$ − select$(S_3, i, 1) - 1 + \delta_{i\geq 2}$. Let $\chi' = \chi_1(S_1) \circ \chi_1(S_2) \circ \chi_1(S_3) \circ \chi_2(S_1)$. From degree$(i)$ = above$(i)$ + below$(i)$ + low$(i)$ + high$(i)$, Fact 5.2, and Lemma 5.3, we determine that degree$(i)$ is computable from $S_1 \circ S_2 \circ S_3 \circ \chi'$ in $O(1)$ time.

To support adjacency queries and listing of all neighbors, we introduce a string $S$ of two types of parentheses derived from $S_1$ and $S_2$ as follows. Although $S$ is only implicitly represented in our convenient encoding, any $O(\log n)$ consecutive parentheses of $S$ can be obtained from $S_1$, $S_2$, and some auxiliary string in $O(1)$ time. Let $\texttt{(}$

and `)` be of type 1, and let `[` and `]` be of type 2. Initially, for each $i = 1, 2, \ldots, 2n$, replace the $i$th 1 of $S_2$ with $S_1[i]$. Then, replace each 0 of $S_2$ with a bracket such that the bracket is open if and only if the last parenthesis in $S$ preceding this 0 is closed. More precisely, for each $i = 1, 2, \ldots, |S_2|$, let

$$S[i] = \begin{cases} S_1[j_1] & \text{if } S_2[i] = 1, \\ \texttt{]} & \text{if } S_2[i] = 0 \text{ and } S_1[j_i] = \texttt{(}, \\ \texttt{[} & \text{if } S_2[i] = 0 \text{ and } S_1[j_i] = \texttt{)}, \end{cases}$$

where $j_i = \operatorname{rank}(S_2, i, 1)$. For example, if $H$ and $T$ are as given in Figure 1.1(a), then $S$ is as in (5.1). There exists an auxiliary string $\chi_3$ such that any $O(\log n)$ consecutive symbols of $S$ are obtainable from $S_1 \circ S_2 \circ \chi_3$ in $O(1)$ time: Let $\ell = \lfloor \frac{1}{8} \log_2 n \rfloor$. Observe that the content of $S[i, i + \ell - 1]$ can be uniquely determined from the concatenation $S'$ of $S_2[i, i + \ell - 1]$ and $S_1[j, j + \ell - 1]$ with $j = \operatorname{rank}(S_2, i, 1)$. Also, $S'$ is obtainable from $S_1 \circ S_2 \circ \chi_1(S_2)$ in $O(1)$ time. Since $S'$ has $4^\ell$ distinct values, we can precompute in $O(n)$ time an $o(n)$-bit table $M$ such that the content of $S[i, i + \ell - 1]$ is obtainable from $S'$ and $M$ in $O(1)$ time. Hence, it suffices to let $\chi_3 = M \circ \chi_1(S_2)$.

With the help of $S$ and its auxiliary strings, adjacency queries can be supported as follows. For any two integers $a$ and $b$, let $[a, b]$ consist of the integers $a, a+1, \ldots, b$. For each $i \in \{1, 2, \ldots, n\}$, let

$$\begin{aligned} L_i &= [\ell_i + 1, \operatorname{select}(S_2, \operatorname{rank}(S_2, \ell_i, 1) + 1, 1) - 1], \\ R_i &= [h_i + 1, \operatorname{select}(S_2, \operatorname{rank}(S_2, h_i, 1) + 1, 1) - 1], \end{aligned}$$

where $\ell_i = \operatorname{select}(S, i, \texttt{(})$ and $h_i = \operatorname{match}(S, \ell_i)$. Let $(v_i, v_j)$ and $(v_{i'}, v_{j'})$, with $i < j$ and $i' < j'$, be two unrelated edges of $H$ with respect to $T$. Since $T$ is an orderly spanning tree of $H$, one can see that if $(v_{i'}, v_{j'})$ is enclosed by the cycle of $H$ determined by $T$ and $(v_i, v_j)$, then $h_i < h_{i'} < \ell_{j'} < \ell_j$. One can prove that

> $v_i$ and $v_j$, with $i < j$, are adjacent in $H - T$ if and only if there exists an index $\ell \in R_i$ with $\operatorname{match}(S, \ell) \in L_j$

by the following induction on the number $b$ of matching bracket pairs in $S$:

> The above statement clearly holds when $b = 0$. To show the induction step for any $b \geq 1$, let $e = (v_x, v_y)$ be an edge in $H - T$. We know that $T$ is also an orderly spanning tree of $H - \{e\}$. Let $\hat{S}$, $\hat{R}_i$, and $\hat{L}_i$ be the corresponding notation for $H - \{e\}$ with respect to $T$. Observe that $\hat{S}$ can be obtained from $S$ by deleting an open bracket in a position in $R_x$ and deleting a closed bracket in a position in $L_y$. Since $v_x$ and $v_y$ are not adjacent in $H - \{e\}$, it follows from the inductive hypothesis that $\operatorname{match}(\hat{S}, \ell) \notin \hat{L}_y$ holds for any index $\ell$ in $\hat{R}_x$; and $\operatorname{match}(\hat{S}, \ell) \notin \hat{R}_x$ holds for any index $\ell$ in $\hat{L}_y$. Therefore, there is a position in $\hat{R}_x$ and a position in $\hat{L}_y$ such that if we insert an open bracket in the first position and a closed bracket in the second position, then the brackets will match each other in the resulting string, which is exactly $S$. Thus, the above statement is proved.

Thus, one can determine whether $(v_i, v_j)$ is an unrelated edge of $H$ with respect to $T$, by checking whether $i'' \in R_i$ and $j'' \in L_j$ hold, where

$$(i'', j'') = \operatorname{enclose}_2(S, \operatorname{select}(S, \operatorname{rank}(S_2, h_i, 1) + 1, \texttt{(}), \ell_j).$$

Therefore, the answer to each adjacency query is derivable from $S_2 \circ S \circ \chi_1(S_2) \circ \chi_1(S)$ in $O(1)$ time.

The neighbors of a degree-$d$ node $v_i$ can be listed from $S \circ \chi_1(S)$ in $O(d)$ time: If $v_i$ is not the root of $T$, then the parent of $v_i$ is $v_j$, where $j$ is computable by letting

$$\begin{aligned}(j_1, j_2) &= \mathrm{enclose}(S, \mathrm{select}(S, i, (), \mathrm{match}(S, \mathrm{select}(S, i, ()));\\ j &= \mathrm{rank}(S, j_1, ().\end{aligned}$$

If $v_i$ is not a leaf of $T$, then $v_{i+1}$ is the first child of $v_i$ in $T$. If $v_j$ is the $t$th child of $v_i$ in $T$, then the $(t+1)$st child of $v_i$ in $T$ is $v_k$, where

$$k = \mathrm{rank}(S, 1 + \mathrm{match}(S, \mathrm{select}(S, j, ()), ().$$

If $t \le |U_<(v_i)|$, the $t$th neighbor of $v_i$ in $U_<(v)$ with respect to $T$ is $v_j$, where $j$ is computable by

$$\begin{aligned}j_1 &= \mathrm{match}(S, t + \mathrm{select}(S, i, ());\\ j_2 &= \mathrm{select}(S, \mathrm{rank}(S, j_1, )), ));\\ j &= \mathrm{rank}(S, \mathrm{match}(S, j_2), ().\end{aligned}$$

If $t \le |D(v_i)|$, then the $t$th neighbor of $v_i$ in $D(v)$ with respect to $T$ is $v_j$, where $j$ is computable by $j_1 = \mathrm{match}(S, \mathrm{select}(S, i, ())$ and $j = \mathrm{rank}(S, \mathrm{match}(S, j_1 + t), ()$.

It is not difficult to verify that $G$ can be reconstructed from $S$ and $S_3$ in $O(m + n)$ time. Therefore, the theorem for connected planar graphs is proved by letting $\chi = \chi' \circ \chi_3 \circ \chi_1(S)$. As for the case that $G$ has $k \ge 2$ connected components, by Theorem 2.3, an orderly pair $(H^i, T^i)$ of the $i$th connected component $G^i$ of $G$ can be derived in overall $O(m + n)$ time. Let $(S_1^i, S_2^i, S_3^i)$ be the $T^i$-code of $H^i$. For each $j = 1, 2, 3$, let $S_j$ be the concatenation of $S_j^1, S_j^2, \ldots, S_j^k$. The theorem can then be proved similarly using $S_1$, $S_2$, and $S_3$. ☐

**Acknowledgments.** We are grateful to the anonymous reviewers for their helpful suggestions which significantly improved the paper. We thank Xin He for helpful comments. We thank Hsu-Chun Yen and Ho-Lin Chen for bringing [16] to our attention. We also thank Richie Chih-Nan Chuang, Yuan-Jiunn Wang, and Kai-Ju Liu for discussions.

## REFERENCES

[1] T. C. Bell, J. G. Cleary, and I. H. Witten, *Text Compression*, Prentice–Hall, Englewood Cliffs, NJ, 1990.

[2] N. Bonichon, C. Gavoille, and N. Hanusse, *An information-theoretic upper bound of planar graphs using triangulation*, in Proceedings of the 20th Annual Symposium on Theoretical Aspects of Computer Science, Lecture Notes in Comput. Sci. 2607, Springer-Verlag, Berlin, 2003, pp. 499–510.

[3] P. Bose, A. M. Dean, and J. P. Hutchinson, *On rectangle visibility graphs*, in Proceedings of the 4th International Symposium on Graph Drawing, Lecture Notes in Comput. Sci. 1190, Springer-Verlag, Berlin, 1996, pp. 25–44.

[4] J. Boyer and W. Myrvold, *Stop minding your p's and q's: A simplified $O(n)$ planar embedding algorithm*, in Proceedings of the 10th Annual ACM-SIAM Symposium on Discrete Algorithms, ACM, New York, SIAM, Philadelphia, 1999, pp. 140–146.

[5] A. Brodnik and J. I. Munro, *Membership in constant time and almost-minimum space*, SIAM J. Comput., 28 (1999), pp. 1627–1640.

[6] H.-L. Chen, C.-C. Liao, H.-I. Lu, and H.-C. Yen, *Some applications of orderly spanning trees in graph drawing*, in Proceedings of the 10th International Symposium on Graph Drawing, Lecture Notes in Comput. Sci. 2528, Springer-Verlag, Berlin, 2002, pp. 332–343.

[7] M. Chrobak and S.-I. Nakano, *Minimum-width grid drawings of plane graphs*, Comput. Geom., 11 (1998), pp. 29–54.

[8] M. Chrobak and T. H. Payne, *A linear-time algorithm for drawing a planar graph on a grid*, Inform. Process. Lett. 54 (1995), pp. 241–246.

[9] R. C.-N. Chuang, A. Garg, X. He, M.-Y. Kao, and H.-I. Lu, *Compact encodings of planar graphs via canonical ordering and multiple parentheses*, in Proceedings of the 25th International Colloquium on Automata, Languages, and Programming, K. G. Larsen, S. Skyum, and G. Winskel, eds., Lecture Notes in Comput. Sci. 1443, Springer-Verlag, Berlin, 1998, pp. 118–129.

[10] D. R. Clark, *Compact PAT Trees*, Ph.D. thesis, University of Waterloo, Ontario, Canada, 1996.

[11] H. de Fraysseix, J. Pach, and R. Pollack, *How to draw a planar graph on a grid*, Combinatorica, 10 (1990), pp. 41–51.

[12] A. M. Dean and J. P. Hutchinson, *Rectangle-visibility representations of bipartite graphs*, Discrete Appl. Math., 75 (1997), pp. 9–25.

[13] A. M. Dean and J. P. Hutchinson, *Rectangle-visibility layouts of unions and products of trees*, J. Graph Algorithms Appl. 2 (1998), pp. 1–21.

[14] B. Dushnik and E. W. Miller, *Partially ordered sets*, Amer. J. Math., 63 (1941), pp. 600–610.

[15] P. Elias, *Universal codeword sets and representations of the integers*, IEEE Trans. Inform. Theory, 21 (1975), pp. 194–203.

[16] U. Fößmeier, G. Kant, and M. Kaufmann, 2-*visibility drawings of planar graphs*, in Proceedings of the 4th International Symposium on Graph Drawing, Lecture Notes in Comput. Sci. 1190, Springer-Verlag, Berlin, 1996, pp. 155–168.

[17] M. L. Fredman and D. E. Willard, *Trans-dichotomous algorithms for minimum spanning trees and shortest paths*, J. Comput. System Sci., 48 (1994), pp. 533–551.

[18] C. Gavoille and N. Hanusse, *Compact routing tables for graphs of bounded genus*, in Proceedings of the 26th International Colloquium on Automata, Languages, and Programming, J. Wiedermann, P. van Emde Boas, and M. Nielsen, eds., Lecture Notes in Comput. Sci. 1644, Springer-Verlag, Berlin, 1999, pp. 351–360.

[19] D. Harel and M. Sardas, *An algorithm for straight-line drawing of planar graphs*, Algorithmica, 20 (1998), pp. 119–135.

[20] X. He, *On floor-plan of plane graphs*, SIAM J. Comput., 28 (1999), pp. 2150–2167.

[21] X. He, M.-Y. Kao, and H.-I. Lu, *A fast general methodology for information-theoretically optimal encodings for graphs*, in Proceedings of the 7th Annual European Symposium on Algorithms, J. Nešetřil, ed., Lecture Notes in Comput. Sci. 1643, Springer-Verlag, Berlin, 1999, pp. 540–549.

[22] X. He, M.-Y. Kao, and H.-I. Lu, *Linear-time succinct encodings of planar graphs via canonical orderings*, SIAM J. Discrete Math., 12 (1999), pp. 317–325.

[23] X. He, M.-Y. Kao, and H.-I. Lu, *A fast general methodology for information-theoretically optimal encodings of graphs*, SIAM J. Comput., 30 (2000), pp. 838–846.

[24] J. Hopcoft and R. E. Tarjan, *Efficient planrity testing*, J. ACM, 21 (1974), pp. 549–568.

[25] J. P. Hutchinson, T. Shermer, and A. Vince, *On representations of some thickness-two graphs*, Comput. Geom., 13 (1999), pp. 161–171.

[26] IEEE, *Proceedings of the* 38*th Annual Symposium on Foundations of Computer Science*, IEEE Computer Society, Los Alamitos, CA, 1997.

[27] G. Kant, *Drawing planar graphs using the canonical ordering*, Algorithmica, 16 (1996), pp. 4–32.

[28] G. Kant and X. He, *Regular edge labeling of* 4*-connected plane graphs and its applications in graph drawing problems*, Theoret. Comput. Sci., 172 (1997), pp. 175–193.

[29] K. Keeler and J. Westbrook, *Short encodings of planar graphs and maps*, Discrete Appl. Math., 58 (1995), pp. 239–252.

[30] C.-C. Liao, H.-I. Lu, and H.-C. Yen, *Floor-planning via orderly spanning trees*, in Proceedings of the 9th International Symposium on Graph Drawing, Lecture Notes in Comput. Sci. 2265, Springer-Verlag, Berlin, 2001, pp. 367–377.

[31] C.-C. Liao, H.-I. Lu, and H.-C. Yen, *Compact floor-planning via orderly spanning trees*, J. Algorithms, 48 (2003), pp. 441–451.

[32] C.-C. Lin, H.-I. Lu, and I.-F. Sun, *Improved compact visibility representation of planar graph via Schnyder's realizer*, SIAM J. Discrete Math., 18 (2004), pp. 19–29.

[33] H.-I. Lu, *Improved compact routing tables for planar networks via orderly spanning trees*, in Proceedings of the 8th Annual International Computing and Combinatorics Conference, Lecture Notes in Comput. Sci. 2387, Springer-Verlag, Berlin, 2002, pp. 57–66.

[34] H.-I. Lu, *Linear-time compression of bounded-genus graphs into information-theoretically optimal number of bits*, in Proceedings of the 13th Annual ACM–SIAM Symposium on Discrete Algorithms, ACM, New York, SIAM, Philadelphia, 2002, pp. 223–224.

[35] J. I. Munro and V. Raman, *Succinct representation of balanced parentheses and static trees*, SIAM J. Comput., 31 (2001), pp. 762–776.

[36] S. North, ed., *Proceedings of the 4th International Symposium on Graph Drawing*, Lecture Notes in Comput. Sci. 1190, Springer-Verlag, Berlin, 1996.

[37] J. Rossignac, *Edgebreaker: Connectivity compression for triangle meshes*, IEEE Trans. Visualization and Computer Graphics, 5 (1999), pp. 47–61.

[38] W. Schnyder, *Planar graphs and poset dimension*, Order, 5 (1989), pp. 323–343.

[39] W. Schnyder, *Embedding planar graphs on the grid*, in Proceedings of the 1st Annual ACM–SIAM Symposium on Discrete Algorithms, ACM, New York, SIAM, Philadelphia, 1990, pp. 138–148.

[40] M. Thorup, *Undirected single source shortest paths in linear time*, in Proceedings of the 38th Annual Symposium on Foundations of Computer Science, IEEE Computer Society, Los Alamitos, CA, 1997, pp. 12–21.

[41] M. Thorup, *On RAM priority queues*, SIAM J. Comput., 30 (2000), pp. 86–109.

[42] W. T. Trotter, *Combinatorics and Partially Ordered Sets—Dimension Theory*, Johns Hopkins University Press, Baltimore, MD, 1992.

[43] G. Turán, *On the succinct representation of graphs*, Discrete Appl. Math., 8 (1984), pp. 289–294.

[44] W. T. Tutte, *A census of planar maps*, Canad. J. Math., 15 (1963), pp. 249–271.

[45] P. van Emde Boas, *Machine models and simulations*, in Handbook of Theoretical Computer Science, Vol. A, J. van Leeuwen, ed., Elsevier, Amsterdam, 1990, pp. 1–60.

[46] M. Yannakakis, *Embedding planar graphs in four pages*, J. Comput. System Sci., 38 (1989), pp. 36–67.

[47] K.-H. Yeap and M. Sarrafzadeh, *Floor-planning by graph dualization: 2-concave rectilinear modules*, SIAM J. Comput., 22 (1993), pp. 500–526.

SIAM J. Comput.
Vol. 34, No. 4, pp. 946–959

# $\Omega(\log n)$ LOWER BOUNDS ON THE AMOUNT OF RANDOMNESS IN 2-PRIVATE COMPUTATION*

ANNA GÁL[†] AND ADI ROSÉN[‡]

**Abstract.** We consider the amount of randomness necessary in information-theoretic private protocols. We prove that at least $\Omega(\log n)$ random bits are necessary for the $t$-private computation of the function xor by $n$ players for any $t \geq 2$. In view of the upper bound of $O(t^2 \log(n/t))$ [E. Kushilevitz and Y. Mansour, *SIAM J. Discrete Math.*, 10 (1997), pp. 647–661], this bound is tight, up to constant factors, for any fixed $t$. For a class of protocols obeying certain restrictions, we give a stronger lower bound of $\Omega(t \log(n/t))$. We note that all known randomness efficient private protocols designed specifically for xor belong to this class. In fact we prove slightly stronger statements: we prove that on *every* input there is a run where the number of random bits used is large, rather than proving only that on some input there is a run where the number of random bits used is large. All our lower bounds hold for the "trusted dealer" model as well, and the $\Omega(t \log(n/t))$ lower bound for restricted protocols is tight, up to constant factors, for any $t \geq 2$ in this model.

In comparison, the previous lower bounds on the amount of randomness required by $t$-private computation of explicit functions did not grow with $n$ for constant values of $t$, and our results improve the previous lower bounds for xor for any $2 \leq t = o(\log n)$. Our results also show that already for $t = 2$, $\Omega(\log n)$ random bits are necessary, while it is known that for the case of $t = 1$ a single random bit is sufficient for privately computing xor for any number of players.

Our proofs use novel techniques by which we extract random variables from a $t$-private protocol, and then use the $t$-privacy property of the protocol to prove properties of these random variables. These properties in turn imply that the number of random bits used by the players is large.

**Key words.** private computation, randomness, lower bounds

**AMS subject classifications.** 68R05, 94A60, 68M10

**DOI.** 10.1137/S0097539703432785

## 1. Introduction.

A *t-private* protocol for computing a function $f$ is a distributed protocol that allows $n$ players $P_i$, $1 \leq i \leq n$, each possessing an individual secret input $x_i$, to compute the value of $f(\vec{x})$ in a way that does not reveal any "unnecessary" information to any coalition of at most $t$ players. The players proceed in rounds, where in each round each player can send a private message to any other player (i.e., each player sends to each other player a message that cannot be seen by any of the remaining players). The $t$-privacy property means that any coalition of at most $t$ players cannot learn anything from the execution of the protocol, except what is implied by the value of $f(\vec{x})$ and the inputs of the members of the coalition. In particular, the members of the coalition do not learn anything about the inputs of the other players. Private computation in this setting was the subject of considerable research; see, e.g., [2, 3, 4, 6, 7, 11, 13, 14, 15, 16, 17, 19, 21, 23, 24, 29]. Randomness is necessary to perform private computations involving more than two players (except for the computation of very degenerate functions). That is, the players must have access to a random source. As randomness is regarded as a scarce

*Received by the editors August 3, 2003; accepted for publication (in revised form) June 28, 2004; published electronically May 12, 2005. A preliminary version of this paper appeared in *Proceedings of the* 35th *Symposium on Theory of Computing (STOC)*, 2003, pp. 659–666.
http://www.siam.org/journals/sicomp/34-4/43278.html

[†]Department of Computer Science, University of Texas at Austin, Austin, TX 78712-1188 (panni@cs.utexas.edu). The work of this author was supported in part by NSF Career Award CCR-9874862 and an Alfred P. Sloan Research Fellowship.

[‡]Department of Computer Science, Technion, Haifa 32000, Israel (adiro@cs.technion.ac.il).

resource, methods for saving random bits in various contexts have been suggested in the literature; see, e.g., [28, 18] for a survey. Thus, an important research topic is the design of randomness-efficient private protocols, and the quantification of the amount of randomness needed to perform private computations of various functions and under various constraints. This line of research has received considerable attention in recent years; see, e.g., [27, 23, 25, 17, 7, 8, 10, 26, 5, 20]. This study also showed that the randomness complexity of the private computation of a function is related to other complexity measures, such as sensitivity and circuit size [27, 25, 17, 7, 26]. The specific function `xor` (addition modulo 2) was the subject of considerable research in this context due to its being a basic operation and its relative simplicity [27, 26, 8, 23].

Previous work on the randomness complexity of private computations revealed that there is a tradeoff between randomness and time (i.e., number of communication rounds) for the 1-private computation of the function `xor` [27, 17]. These works also gave lower bounds on the number of rounds necessary to 1-privately compute any function in terms of the sensitivity of the function and the amount of randomness used. If one is allowed an arbitrary number of rounds for the computation, there are no known lower bounds on the number of random bits for 1-private protocols computing explicit functions (except that randomness is necessary, i.e., no deterministic private protocol exists). In fact, Kushilevitz, Ostrovsky, and Rosén [25] gave a relation between the number of random bits necessary to 1-privately compute a function and the Boolean circuit size necessary to compute it; it is proved that the class of functions that have $O(1)$-random, 1-private protocols is equal to the class of functions that have linear-size circuits. This surprising connection explains the lack of $\omega(1)$ lower bounds on the number of random bits for explicit functions in the case of 1-privacy, as such results would imply superlinear lower bounds on circuit size.

Before our work, $\omega(1)$ lower bounds on the number of random bits of $t$-private protocols (without limiting the number of rounds) have been proved for explicit functions only for values of $t$ that grow with $n$, and no such bounds have been known if $t$ itself is constant. More precisely, Kushilevitz and Mansour [23] proved that any $t$-private protocol for `xor` requires at least $t$ random bits. Blundo et al. [7] gave lower bounds for two special cases. Namely, they proved that if $t = n - c$ for some constant $c$, then $\Omega(n^2)$ random bits are necessary, and if $t \geq (2 - \sqrt{2})n$, then $\Omega(n)$ random bits are necessary. As for upper bounds, Canetti et al. [10] gave randomness-efficient generic protocols to $t$-privately compute (for $t < n/2$) any Boolean function $f$. They showed that any function $f$ with circuit size of $m$ gates can be computed by a $t$-private protocol ($t < n/2$) using $O(t^2 \log n + (m/n)t^5 \log t)$ random bits. Kushilevitz and Mansour [23] gave protocols that compute the function `xor` $t$-privately, for any $t$, using $O(t^2 \log(n/t))$ random bits.

In the present paper we develop new techniques for proving lower bounds on the number of random bits necessary in $t$-private computations (for $t \geq 2$) and obtain $\Omega(\log n)$ lower bounds on the number of random bits necessary to $t$-privately compute the function `xor` for any $t \geq 2$.[1] More precisely, we prove the following theorem. (See section 2 for a formal definition of a $d$-random protocol.)

THEOREM 1.1. *Let $t \geq 2$, and let $\mathcal{A}$ be a $d$-random, $t$-private protocol for computing $f(\vec{x}) = x_1 + \cdots + x_n \pmod 2$. Then $d = \Omega(\log n)$.*

In fact, we prove a slightly stronger statement: we prove that $\Omega(\log n)$ random bits are necessary on *every* input.

[1]Blundo, Galdi, and Persiano [9] recently reported obtaining similar results independently, using a different approach.

In view of the upper bound of $O(t^2 \log(n/t))$ of [23], our lower bound is tight, up to constant factors, for any fixed $t$. This is the first result showing that the number of random bits necessary for $t$-private computation grows with $n$ for constant values of $t$, and it improves the lower bound of [23] for any $t = o(\log n)$. It is interesting to note that our $\Omega(\log n)$ lower bound holds already for $t = 2$, while for the case of $t = 1$, it is known that the function `xor` can be computed 1-privately, for any number of players $n$, with only 1 random bit.

All known randomness-efficient private protocols designed specifically for the function `xor` [27, 23, 26, 8] are built in the following special way. They are based on a deterministic, nonprivate protocol for `xor`. Then this protocol is modified by changing any message so it is the sum (modulo 2) of the original message, and a value which is a function of the random bits only. Thus, the private protocol is built by masking the original messages of the nonprivate protocol. We give stronger lower bounds for protocols of this class (see section 4 for a formal definition of this class). Namely, we give a lower bound of $\Omega(t \log(n/t))$ on the number of random bits required by any protocol of this class to compute `xor` for $n$ players.

All our lower bounds hold also in the "trusted dealer" model, considered in [23, 10]. In this model, the $n$ players are deterministic, and there is an additional player, the "trusted dealer," who does not get any input, and whose role is limited to "deal" random bits to the other players (hence a "dealer"). This player never participates in any coalition (hence it is "trusted"). For this model, our lower bound of $\Omega(t \log(n/t))$ for protocols of the above restricted class is tight up to constant factors for every $t \geq 2$, as [23] gave a protocol (of this class) in the trusted dealer model using $O(t \log(n/t))$ random bits.

Our proofs use novel techniques by which we extract from a private protocol random variables that depend on the randomness that the players use. We then use the $t$-privacy property of the protocol to prove that these random variables must have certain properties (for example, linear independence or $t$-wise independence). Based on these properties we show that the amount of randomness used by the players must be large. We believe that these new techniques may prove useful for proving other properties of private protocols.

## 2. Preliminaries.

In this paper we consider information-theoretic privacy (as in [4, 11]), where the players have unlimited computational power, no intractability assumptions are made, and messages are sent over private channels.

Let $f : \{0,1\}^n \to \{0,1\}$ be an arbitrary Boolean function. A set of $n$ players $P_i$ ($1 \leq i \leq n$), each possessing a single private input bit $x_i$ (i.e., $x_i$ is known *only* to $P_i$), collaborate in a protocol to compute the value of $f(\vec{x})$. The protocol is probabilistic. During the course of the protocol each player can toss random coins, where the coin tosses are unbiased and independent. The protocol operates in rounds. In each round, each player may toss some coins, and then sends messages to the other players (messages are sent over private channels so that, other than the intended receiver, no other player can access them). The player then receives the messages sent to it by the other players. Each player chooses to output the value of the function at a certain round. In a correct protocol, each player must output the correct value $f(\vec{x})$, and stop its operation in a finite number of steps (it may output $f(\vec{x})$ before stopping). That is, for every input assignment $\vec{x}$ and for every outcome of the coin tosses of all players, each player outputs $f(\vec{x})$ and stops in a finite number of steps.

In Claim 1 below we formally argue that for a given correct protocol involving $n$ players, there is a finite upper bound $\ell$ on the number of coin tosses any single player performs during the course of the protocol. We in fact show that there is a finite upper bound $\ell$ on the total number of coin tosses performed by all players. Claim 1 gives a formal argument for the intuition that a protocol for which such an upper bound does not exist is not a correct protocol. For example, consider a protocol where some player keeps tossing coins until it gets a 1 before sending any message. For such a protocol, the property claimed does not hold. But such a protocol does not satisfy the definition of correctness either, as there are possible outcomes of the coin tosses for which some player does not stop in a finite number of steps. We note that since we prove lower bounds on the number of random bits used, we could avoid using Claim 1 by the following argument: we could argue that if on some input, some player may toss more than $\ell$ coins, then a lower bound of $\ell$ is obtained; otherwise one can assume that no player ever tosses more than $\ell$ coins. This would be sufficient to prove lower bounds on the randomness complexity of the protocol (see Definition 2.1). However, we prove stronger statements. Claim 1 is useful in proving that on *every* input there is a run where the number of coin tosses performed is large, rather than only proving that on some input there is a run where the number of coin tosses performed is large.

CLAIM 1. *Given a correct protocol involving $n$ players, there is a finite upper bound $\ell$ on the total number of coin tosses performed by all players in any run of the protocol.*

*Proof.* We show below that for any input assignment $\vec{x}$ there is a finite upper bound $\ell(\vec{x})$ on the total number of coin tosses performed by all players in any run of the protocol in which the input assignment is $\vec{x}$. Since there is a finite number of input assignments $\vec{x} \in \{0,1\}^n$, the claim follows by letting $\ell = \max_{\vec{x}\in\{0,1\}^n}\{\ell(\vec{x})\}$.

Fix an input assignment $\vec{x} \in \{0,1\}^n$. As in the proof of Lemma 4.10 in [27], we build a binary tree $T_{\vec{x}}$ representing the coin tosses of the players on a given input $\vec{x}$. Each node of the tree is labeled by the name of a player $P_i$, which tosses a coin. The two outgoing edges from a node are labeled 0 and 1 according to the outcome of the coin toss. Coin tosses in the run of the protocol are ordered by round number, then by player number, and then by a serial number (for that player in that round). Note that the identity of the player to toss the first coin on $\vec{x}$, that is, the label of the root, is determined by $\vec{x}$, and the identity of any subsequent player to toss a coin is determined by $\vec{x}$ and the outcomes of the previous coin tosses, that is, by the path leading to a given node of the tree.

Observe that a path of length $k$ from the root to another node represents a run (or a prefix of a run) of the protocol in which $k$ coin tosses occur. Assume towards a contradiction that there is no finite upper bound $\ell(\vec{x})$ on the number of coin tosses performed by the players on input $\vec{x}$. Then for every finite $k$ there is a path of length $k$ in the tree. Since the outdegree of each node of the tree is at most 2, this means that the tree $T_{\vec{x}}$ must contain an infinite path starting from the root (cf. König's lemma in [22]). This path corresponds to a possible run of the protocol (defined by $\vec{x}$ and the results of the coin tosses as defined by the edges along the path). In this run at least one player tosses an infinite number of coins, i.e., this player does not stop in a finite number of steps, contradicting the correctness of the protocol. ∎

We thus model the players in a correct protocol as being provided with finite binary random tapes. That is, in a correct protocol involving $n$ players to compute a function $f$, each player $P_i$ is provided with a local binary random tape $R_i$ of length $\ell$. Note that the value of $\ell$ may be different for different protocols and different numbers of players $n$. The bits in the random tapes are unbiased and independent. We denote by $\vec{R} = (R_1, \ldots, R_n)$ a given vector of random tapes of all players, and we think of $\vec{R}$ as a binary vector of length $n\ell$.

The following definition is used to measure the amount of randomness used in a protocol.

Definition 2.1 (randomness complexity of a protocol). *A $d$-random protocol is a protocol such that for any input assignment and any vector of local random tapes, the total number of random bits read from the local random tapes by all players is at most $d$.*

We emphasize that the definitions allow, for example, that in different executions of a protocol (i.e., different input assignments and different local random tapes), a given player reads a different number of random bits from its local random tape. The number of random bits read by the player may depend on both the inputs of the players and the random bits read by all players.

We now proceed to consider the messages exchanged by the players. Each player $P_i$ receives during the execution of the protocol a sequence of messages. In different runs of a protocol the various players may receive different messages. These depend on the input to the players and on the random tapes. We denote the communication seen by a player as follows.

Definition 2.2. *The* communication $c_i(\vec{x}, \vec{R})$, *of player $P_i$, on input $\vec{x}$ and vector of random tapes $\vec{R} = \{R_i\}_{1 \le i \le n}$, is the sequence of messages that player $P_i$ receives during the execution of the protocol when the input is $\vec{x}$ and the vector of random tapes of all players is $\vec{R}$.*

Thus, $c_i$ is the (random) variable of the sequence of messages received by $P_i$. For a subset of the players $S$, we denote by $c_S$ the (random) variable of the sequences of messages received by all the players in $S$. Informally, *$t$-privacy* means that any coalition of up to $t$ players cannot learn anything (in particular, the inputs of the other players) from the communication that the members of the coalition receive, except what is implied by the input bits of the members of that coalition, and the value of the function computed. Formally, we give the following definition.

Definition 2.3 (privacy). *A protocol for computing a function $f$ is private with respect to a subset of the players $S \subseteq [n]$ if the following holds. For any two input vectors $\vec{x}$ and $\vec{y}$ such that $f(\vec{x}) = f(\vec{y})$, and $x_i = y_i$ for any $i \in S$, and for any sequence of messages $C_S$, and for any vector of random tapes for the subset $S$, $\{R_i\}_{i \in S}$,*

$$\Pr[c_S = C_S | \{R_i\}_{i \in S}, \vec{x}\,] = \Pr[c_S = C_S | \{R_i\}_{i \in S}, \vec{y}\,] \ ,$$

*where the probability is over the random tapes of the players.*

A protocol is said to be $t$-private if it is private with respect to any subset of players $S$ such that $|S| \le t$.

It will be convenient in our proofs to use a weaker privacy requirement, directly implied by the $t$-privacy property, as stated in the following lemma. (Note that since we prove lower bounds, this only makes our results stronger.)

Lemma 2.4. *Consider any $t$-private protocol. For any subset $S$ of the players $S \subseteq [n]$ such that $|S| \le t$, and for any two input vectors $\vec{x}$ and $\vec{y}$ such that $f(\vec{x}) = f(\vec{y})$ and $x_i = y_i$ for any $i \in S$, the following holds:*

1. *For any sequence of messages $C_S$,*

$$\Pr[c_S = C_S | \vec{x}\,] = \Pr[c_S = C_S | \vec{y}\,] \ ,$$

*where the probability is over the vectors $\vec{R}$ chosen uniformly from $\{0,1\}^{n\ell}$.*

2. *For any function $\phi^S$ of $c_S$, and for any value $\Phi$ in the range of $\phi^S$,*

$$\Pr[\phi^S = \Phi | \vec{x}\,] = \Pr[\phi^S = \Phi | \vec{y}\,] \ ,$$

*where the probability is over the vectors $\vec{R}$ chosen uniformly from $\{0,1\}^{n\ell}$.*

*Proof.* Let $s$ be the size of $S$, i.e., $s = |S|$. Fixing a vector of random tapes for the subset $S$, $\{R_i\}_{i \in S}$, is equivalent to fixing a binary vector of length $s\ell$. The probability of each of these $2^{s\ell}$ vectors is $2^{-s\ell}$, and the events corresponding to the various vectors are disjoint. Therefore we have

$$\Pr[c_S = C_S | \vec{x}\,] = 2^{-s\ell} \sum_{\{R_i\}_{i \in S} \in \{0,1\}^{s\ell}} \Pr[c_S = C_S | \{R_i\}_{i \in S}, \vec{x}\,] \ .$$

Using the same arguments, applied to $\vec{y}$ instead of $\vec{x}$, we have that

$$\Pr[c_S = C_S | \vec{y}\,] = 2^{-s\ell} \sum_{\{R_i\}_{i \in S} \in \{0,1\}^{s\ell}} \Pr[c_S = C_S | \{R_i\}_{i \in S}, \vec{y}\,] \ .$$

But, by the privacy property of the protocol we have that for any vector of random tapes for the subset $S$, $\{R_i\}_{i \in S}$,

$$\Pr[c_S = C_S | \{R_i\}_{i \in S}, \vec{x}\,] = \Pr[c_S = C_S | \{R_i\}_{i \in S}, \vec{y}\,] \ .$$

We therefore obtain that

$$\Pr[c_S = C_S | \vec{x}\,] = \Pr[c_S = C_S | \vec{y}\,] \ .$$

The second statement of the lemma follows by observing that the value of $\phi^S$ is fixed given any communication $C_S$. □

From the point of view of an observer of the protocol, one can define the *transcript* of a given run of the protocol, which is the sequence of all messages sent between all players during the execution of the protocol on input $\vec{x}$ and vector of random tapes $\vec{R}$. The transcript is in fact the ordered vector of the communication of all players.

Definition 2.5. *The* transcript $Trans(\vec{x}, \vec{R})$ *of a protocol on input $\vec{x}$ and vector of random tapes $\vec{R} = \{R_i\}_{1 \leq i \leq n}$ is $(c_1(\vec{x}, \vec{R}), c_2(\vec{x}, \vec{R}), \ldots, c_n(\vec{x}, \vec{R}))$.*

The following lemma follows immediately from the arguments of the proof of Lemma 4.10 in [27]. We will use this lemma in our proofs.

Lemma 2.6 (see [27]). *For a given input $\vec{x}$, let $d$ be the maximum, over all runs on input $\vec{x}$, of the total number of random bits read from the random tapes by all players during a given run. Then the number of different transcripts of runs on input $\vec{x}$ is at most $2^d$.*

It is convenient in our proofs to consider the messages sent by the players as being messages of single bits. This is done by "breaking" each message into the bits of its binary representation. Formally, for a given protocol involving $n$ players, let $M$ be the set of all different messages that can be sent in the protocol in all different runs (over all possible inputs $\vec{x} \in \{0,1\}^n$ and all possible vectors of random tapes $\vec{R} \in \{0,1\}^{n\ell}$). Fix an arbitrary one-to-one binary encoding of fixed length for the messages in $M$. We note that the empty message is one of the elements of $M$. We consider a protocol where each player sends, instead of a given message from $M$, a sequence of single-bit messages that represents the binary encoding of the original message from $M$. Henceforth, when we refer to *messages* we refer to these single-bit messages. It is important for our argument that the number of transcripts of the protocol, on any given input $\vec{x}$, remains the same. This follows since we use a one-to-one encoding.

Since we consider each message as being a single bit, we can think of a given message $m$ as a Boolean function of the input $\vec{x}$, which is a binary vector of length $n$, and the random tapes of all players, which is a binary vector of length $n\ell$. We therefore can write $m$ as $m = m(\vec{x}, \vec{R})$.

Our lower bound exploits the fact that the function xor has large sensitivity on every input. Sensitivity is defined as follows.

DEFINITION 2.7 (sensitivity).

- *Given $\vec{x} \in \{0,1\}^n$, we denote by $\vec{x}^{(i)}$ the vector $\vec{x}$ with its ith bit flipped. (Similarly, $\vec{x}^{(i,j)}$ is $\vec{x}$ with its ith and jth bits flipped.)*
- *A function $f$ is* sensitive *to its ith variable on input $\vec{x}$ if $f(\vec{x}) \neq f(\vec{x}^{(i)})$.*
- *$s(f, \vec{x})$ is the number of variables to which the function $f$ is sensitive on input $\vec{x}$.*
- *The* sensitivity *of a function $f$ is $s(f) = \max_{\vec{x}} s(f, \vec{x})$.*

Note that the function xor (addition modulo 2) of $n$ binary variables is sensitive to all its $n$ variables on any input. This immediately follows since for any $\vec{x} \in \{0,1\}^n$, and for any $i \in [n]$, $\texttt{xor}(\vec{x}) \neq \texttt{xor}(\vec{x}^{(i)})$.

We will further need the following definitions.

DEFINITION 2.8.

- *A message $m$ depends on the variable $x_i$ if there exist $\vec{x}$ and $\vec{R}$, such that $m(\vec{x}, \vec{R}) \neq m(\vec{x}^{(i)}, \vec{R})$. In other words, $m$ depends on the variable $x_i$ if $m$ is sensitive to $x_i$ on some $\vec{x}$ and $\vec{R}$.*
- *For $i \leq j$, a message $m$ depends on a variable $x_j$ under the partial assignment $x_1 = \alpha_1, \ldots, x_{i-1} = \alpha_{i-1}$ if there exists an assignment to the remaining variables $x_i, \ldots, x_n$ and there exists $\vec{R}$, such that $m(\alpha_1, \ldots, \alpha_{i-1}, x_i, \ldots, x_j, \ldots, x_n, \vec{R}) \neq m(\alpha_1, \ldots, \alpha_{i-1}, x_i, \ldots, \bar{x_j}, \ldots, x_n, \vec{R})$.*

We will use the following simple observation.

OBSERVATION 1. *Let $m = m(\vec{x}, \vec{R}) = \phi(f_1(\vec{x}, \vec{R}), \ldots, f_u(\vec{x}, \vec{R}))$. If $m$ depends on a variable $x_j$ under the partial assignment $x_1 = \alpha_1, \ldots, x_{i-1} = \alpha_{i-1}$, then at least one of the functions $f_1, \ldots, f_u$ depends on the variable $x_j$ under the partial assignment $x_1 = \alpha_1, \ldots, x_{i-1} = \alpha_{i-1}$.*

## 3. Lower bound for general protocols.

In this section we give a lower bound that applies to any $t$-private protocol for xor for $t \geq 2$. We first outline our approach, which is common to the general lower bound and to the stronger lower bound for the restricted class of protocols (given in section 4) and then proceed to give the proof of Theorem 1.1.

### 3.1. Our approach.

We state informally the approach we use in our proofs. Our proofs proceed in two stages. First, we prove that in any $t$-private protocol for xor we can identify $q = \Omega(n)$ distinct messages $m_1, \ldots, m_q$ with certain properties. Informally these properties are as follows:

(1) We can permute the input vector (and accordingly the set of players), such that for any $i$, message $m_i$ depends on input $x_i$ but does not depend on any input $x_j$, $j > i$.

(2) The set of receivers of these messages is disjoint from the set of players that have access to the inputs $x_i$, $1 \leq i \leq q$.

In the second stage of our proofs we consider the values of these selected messages on a given input assignment. That is, we consider the vectors representing the values of these messages over the possible random tapes to all players, when the input assignment is fixed. Using the properties of the private protocol and the properties of the special set of selected messages, we then prove, in the case of a general $t$-private protocol, that these vectors are linearly independent. In the case of a protocol of the restricted class, we prove that all the vectors obtained from sums of at most $t/2$ original vectors are linearly independent. In each case, this allows us to conclude that the number of different columns in the matrix obtained from the vectors as rows is "large" (where the extent to which this number is large is different in each case). It follows that the number of transcripts of the protocol on the given input is "large," and hence, using Lemma 2.6, the randomness complexity of the protocol is "high."

**3.2. Proof of Theorem 1.1.** In this section, as well as in section 4, we always assume that the computed function is `xor`. We now proceed to prove Theorem 1.1. In fact we prove here a stronger claim than the claim of Theorem 1.1. We prove that for any input assignment $\vec{\alpha} = \alpha_1, \ldots, \alpha_n$, there is a run in which the number of random bits read by the players from their random tapes is $\Omega(\log n)$.

**3.2.1. Selecting the messages.** Let $\vec{\alpha} = \alpha_1, \ldots, \alpha_n$ be an arbitrary input assignment. Given a fixed $\vec{\alpha}$, we will define a sequence of messages. (We will not indicate in our notation that the choice of this sequence depends on $\vec{\alpha}$, but this should be clear from the context.)

We define an ordering of all the messages sent during the protocol in order to be able to refer to *the first* message with a given property. Then, based on this ordering, we select a sequence of messages with certain properties. The choice of these messages will induce a particular permutation of the input bits; and since each input bit belongs to a given player, this induces a permutation of the players as well.

DEFINITION 3.1. *We define an ordering of all messages, such that in this ordering, any message sent in round $i$ precedes any message sent in round $j$ for $i < j$. For the messages sent within the same round we choose an arbitrary ordering.*

When we refer to the first message with a given property, we mean the first message according to the above ordering that satisfies that property.

During the process of selecting the sequence of messages, we also assign a particular numbering to the input bits and to the players. To this end, when selecting a given message, a variable and a player are also selected, and both are given the same number as the message. That is, when the first $i$ messages $m_1, \ldots, m_i$ have been selected, the variables $x_1, \ldots, x_i$ and the players $P_1, \ldots, P_i$ are also already selected. (We assume an arbitrary permutation of the remaining indices $i + 1, \ldots, n$ for the remaining variables and players.) When the process ends (after selecting $n$ messages) a permutation of the variables and a permutation of the players is fixed.

We now proceed to the selection process. Let $m_1$ be the first message in the protocol that depends on at least one input variable. We will argue below that since this is the first such message, it can depend on only one input variable, and without loss of generality, we denote this input variable by $x_1$, and the player that has access to it by $P_1$.

Let $m_2$ be the first message in the protocol that depends on at least one input variable under $x_1 = \alpha_1$. We will argue below that since this is the first such message, there is only one such input variable, and without loss of generality we denote it by $x_2$, and the player that has access to it by $P_2$.

Inductively, let $m_i$ be the first message sent in the protocol that depends on some input variable under $x_1 = \alpha_1, \ldots, x_{i-1} = \alpha_{i-1}$. We prove the following claim.

CLAIM 2. *Let $x_k$ be any input variable on which $m_i$ depends under $x_1 = \alpha_1, \ldots, x_{i-1} = \alpha_{i-1}$. Then the sender of the message $m_i$ must be the player that has access to the variable $x_k$, that is, player $P_k$.*

*Proof.* Suppose that the sender of the message $m_i$ is a player $P_j$ such that $j \neq k$ (i.e., $P_j$ is not the owner of $x_k$). Note that a message $m$ sent in a given round by player $P_j$ is a function of only the communication to player $P_j$ in previous rounds, its input $x_j$, and its random tape $R_j$. Thus, by Observation 1, if $P_j$ is not the owner of the variable $x_k$, it can send a message that depends on $x_k$ under $x_1 = \alpha_1, \ldots, x_{i-1} = \alpha_{i-1}$ only if it received in an earlier round a message that depends on $x_k$ under $x_1 = \alpha_1, \ldots, x_{i-1} = \alpha_{i-1}$. But this contradicts the assumption that $m_i$ is the first such message. □

The above claim implies that there is only one input variable on which $m_i$ depends under $x_1 = \alpha_1, \ldots, x_{i-1} = \alpha_{i-1}$. Without loss of generality we denote it by $x_i$, and the player that has access to it by $P_i$. Thus we derive the following.

CLAIM 3.

1. *The message $m_i$ is sensitive to $x_i$ on the input $\vec{\alpha}$ and some vector of random tapes $\vec{R}$.*
2. *Let $\vec{\beta}$ be any input that agrees with $\vec{\alpha}$ in the first $i-1$ coordinates, and let $j > i$. Then the message $m_i$ is not sensitive to the variable $x_j$ on $\vec{\beta}$ and $\vec{R}$ for any $\vec{R}$.*

*Proof.* We selected $m_i$ such that $m_i$ depends on $x_i$ under $x_1 = \alpha_1, \ldots, x_{i-1} = \alpha_{i-1}$. This means that there is some assignment to the remaining variables and some $\vec{R}$ such that $m_i$ is sensitive to $x_i$ on the input obtained by the additional assignment and $\vec{R}$. But since $x_i$ is the only input variable on which $m_i$ depends under $x_1 = \alpha_1, \ldots, x_{i-1} = \alpha_{i-1}$ this also means that $m_i$ is sensitive to $x_i$ on $\vec{\alpha}$ and $\vec{R}$.

We obtain the second statement using the observation that $x_i$ is the only input variable on which $m_i$ depends under $x_1 = \alpha_1, \ldots, x_{i-1} = \alpha_{i-1}$. □

CLAIM 4. *We can continue the above procedure of selecting messages for $n$ steps and define the sequence of messages $m_1, \ldots, m_n$.*

*Proof.* In a correct protocol to compute the function $f$, the output of each player has to be equal to $f(\vec{\alpha})$ on any input $\vec{\alpha}$. Note that the output of a given player depends only on the communication it received, its input bit, and its random tape. Since the sensitivity $s(f, \vec{\alpha})$ of the function $f(\vec{x}) = x_1 + \cdots + x_n \pmod 2$ is $n$ on every input $\vec{\alpha}$, we have by Observation 1, for each player $P_i$ and each variable $x_j$ such that $j \neq i$, that the communication received by $P_i$ must contain at least one message that is sensitive to $x_j$ on the input $\vec{\alpha}$ and some $\vec{R}$. Thus, on any input $\vec{\alpha}$, there exists at least one message for each variable $x_j$ that is sensitive to $x_j$ on $\vec{\alpha}$ and some $\vec{R}$. If our procedure cannot be continued after $k < n$ steps on some input $\vec{\alpha}$, that would mean by Claim 3 that no message is sensitive to any of the remaining variables on $\vec{\alpha}$ and $\vec{R}$ for any $\vec{R}$, which would be a contradiction. □

As argued above, the senders of the messages $m_1, \ldots, m_n$ are $P_1, \ldots, P_n$, respectively. Denote by $Q_1, \ldots, Q_n$ the receivers of these messages. Note that the $n$ receivers are not necessarily $n$ distinct players. We now select a subset of the above $n$ messages, $m_{i_1}, \ldots, m_{i_q}$, with the property that $\{P_{i_j} : j \in [q]\} \cap \{Q_{i_j} : j \in [q]\} = \emptyset$. That is, none of the receivers of the selected messages is a sender of a selected message.

LEMMA 3.2. *There is a subset of size $q \geq \frac{n}{4}$ of the above $n$ messages, denoted $m_{i_1}, \ldots, m_{i_q}$, such that the receivers of these messages, $Q_{i_1}, \ldots, Q_{i_q}$ are disjoint from the senders of these messages, $P_{i_1}, \ldots, P_{i_q}$.*

*Proof.* For the purpose of the proof we define an undirected graph. The set of nodes consists of $n$ nodes, each node $v_i$ representing a message $m_i$ of the original set of $n$ messages. Recall that each distinct message $m_i$ is sent by a distinct player $P_i$. Therefore we can also think of the nodes as representing $n$ distinct players. For each message $m_i$, we put an edge between node $v_i$ and node $v_j$ if player $P_j$ is the receiver of message $m_i$.

We now have a graph of $n$ nodes and at most $n$ edges. The graph therefore contains an independent set of size at least $\frac{n}{4}$ (cf. [1, Theorem 3.2.1]). We select the messages that correspond to the nodes of this independent set. $\square$

To simplify notation, in what follows we denote by $m_1, \ldots, m_q$, $P_1, \ldots, P_q$, $x_1, \ldots, x_q$, and $Q_1, \ldots, Q_q$ the selected messages, their senders, the input variables these senders have access to, and the receivers of the messages, respectively.

**3.2.2. Properties of the vectors defined by the selected messages.** We will now consider the $2^{n\ell}$-bit binary vectors that represent these messages on input $\vec{\alpha}$. We denote by $\vec{m}(\vec{\alpha})$ the binary vector of length $2^{n\ell}$ that consists of the bits $m(\vec{\alpha}, \vec{R})$. Thus, for any $i$, the vector $\vec{m}_i(\vec{\alpha})$ consists of the bits $m_i(\vec{\alpha}, \vec{R})$. For $\emptyset \neq S \subseteq [q]$ we denote by $\vec{m}_S(\vec{\alpha})$ the bitwise mod 2 sum of the vectors $\vec{m}_i(\vec{\alpha})$ for $i \in S$. That is, $m_S(\vec{\alpha}, \vec{R}) = \oplus_{i \in S} m_i(\vec{\alpha}, \vec{R})$.

LEMMA 3.3. *Let $m_1, \ldots, m_q$ be selected as above in a $t$-private protocol. For any $\emptyset \neq S \subseteq [q]$ of size at most $t$, and any $i, j \in [q]$ (where $i \neq j$),*

$$\Pr_{\vec{R}}[m_S(\vec{\alpha}, \vec{R}) = 1] = \Pr_{\vec{R}}[m_S(\vec{\alpha}^{(i,j)}, \vec{R}) = 1] .$$

*Proof.* Consider the set of players $S' = \{Q_i | i \in S\}$, that is, the coalition formed by the receivers of the messages $m_i$ for $i \in S$. Then $\phi^{S'}(c_{S'}) = \oplus_{i \in S} m_i(\vec{\alpha}, \vec{R}) = m_S(\vec{\alpha}, \vec{R})$ is a function of the sequence of messages received by the players in $S'$. Recall that the set of senders of the messages $m_i$ is disjoint from the set of the receivers, that is, $\{P_i : i \in [q]\} \cap \{Q_i : i \in [q]\} = \emptyset$, which implies that $\{P_i : i \in [q]\} \cap S' = \emptyset$, and therefore for any $i, j \in [q]$, $P_i$ and $P_j$ are not in $S'$. This means that for any $i, j \in [q]$ and for any $l \in S'$, $\alpha_l = \alpha_l^{(i,j)}$, that is, the input bits held by the members of the coalition $S'$ are not changed when one flips the $i$th and $j$th bits of $\vec{\alpha}$. Note also that flipping two bits does not change the value of the `xor` function, that is, $f(\vec{\alpha}) = f(\vec{\alpha}^{(i,j)})$ for any $i, j \in [q]$, when $f$ is the `xor` function. Thus, we can apply Lemma 2.4 to $S'$ and $\phi^{S'} = m_S$, and the second statement of Lemma 2.4 directly implies the statement of the present lemma. $\square$

For $i = 1, \ldots, q$, we denote by $\vec{h}_i(\vec{\alpha})$ the bitwise mod2 sum of the vectors $\vec{m}_i(\vec{\alpha})$ and $\vec{m}_i(\vec{\alpha}^{(i)})$. Thus, the vector $\vec{h}_i(\vec{\alpha})$ is 1 in the coordinates corresponding to $\vec{R}$ such that $m_i(\vec{\alpha}, \vec{R})$ and $m_i(\vec{\alpha}^{(i)}, \vec{R})$ differ, and 0 where they agree. We denote by $h_i(\vec{\alpha}, \vec{R})$ the entry of $\vec{h}_i(\vec{\alpha})$ in the coordinate corresponding to $\vec{R}$.

CLAIM 5. *The vectors $\vec{h}_i(\vec{\alpha})$, $i = 1, \ldots, q$, are not identically* 0.

*Proof.* The proof follows by the definition of the messages $m_i$ and the first statement of Claim 3. $\square$

LEMMA 3.4. *Let $m_1, \ldots, m_q$ be selected as above in a $t$-private protocol. Let $\emptyset \neq S \subseteq [q-1]$ be a subset of size at most $t$, and let $k$ be the largest element of $S$. Then*

$$\Pr_{\vec{R}}[m_S(\vec{\alpha}, \vec{R}) = 1 | h_k(\vec{\alpha}, \vec{R}) = 1] = 1/2 .$$

*Proof.* Since $k$ is the largest element of $S$ and $q$ is larger than any element in $S$, we get by Claim 3 that $\vec{m}_S(\vec{\alpha}^{(k,q)})$ is the bitwise mod 2 sum of the vectors $\vec{m}_S(\vec{\alpha})$ and $\vec{h}_k(\vec{\alpha})$. To see this observe that for any $1 \leq i < k$, $m_i(\vec{\alpha}^{(k)}, \vec{R}) = m_i(\vec{\alpha}, \vec{R})$ by the second statement of Claim 3, and $m_i(\vec{\alpha}^{(k,q)}, \vec{R}) = m_i(\vec{\alpha}^{(k)}, \vec{R})$ again by the second statement of Claim 3. At the same time, $m_k(\vec{\alpha}^{(k,q)}, \vec{R}) = m_k(\vec{\alpha}^{(k)}, \vec{R})$ by the second statement of Claim 3, and $m_k(\vec{\alpha}^{(k)}, \vec{R}) = m_k(\vec{\alpha}, \vec{R}) + h_k(\vec{\alpha}, \vec{R})$ by the definition of $h_k$. Thus, $m_S(\vec{\alpha}^{(k,q)}, \vec{R})$ and $m_S(\vec{\alpha}, \vec{R})$ are complements of each other in the coordinates where $h_k(\vec{\alpha}, \vec{R})$ is 1, and agree where $h_k(\vec{\alpha}, \vec{R})$ is 0. We therefore have

$$\begin{aligned}\Pr_{\vec{R}}[m_S(\vec{\alpha}, \vec{R}) = 1] = \Pr_{\vec{R}}&[m_S(\vec{\alpha}, \vec{R}) = 1 \wedge h_k(\vec{\alpha}, \vec{R}) = 1] \\ &+ \Pr_{\vec{R}}[m_S(\vec{\alpha}, \vec{R}) = 1 \wedge h_k(\vec{\alpha}, \vec{R}) = 0]\end{aligned}$$

and

$$\begin{aligned}\Pr_{\vec{R}}[m_S(\vec{\alpha}^{(k,q)}, \vec{R}) = 1] = \Pr_{\vec{R}}&[m_S(\vec{\alpha}, \vec{R}) = 0 \wedge h_k(\vec{\alpha}, \vec{R}) = 1] \\ &+ \Pr_{\vec{R}}[m_S(\vec{\alpha}, \vec{R}) = 1 \wedge h_k(\vec{\alpha}, \vec{R}) = 0] \ .\end{aligned}$$

Since by Lemma 3.3

$$\Pr_{\vec{R}}[m_S(\vec{\alpha}, \vec{R}) = 1] = \Pr_{\vec{R}}[m_S(\vec{\alpha}^{(k,q)}, \vec{R}) = 1] \ ,$$

we have that

$$\Pr_{\vec{R}}[m_S(\vec{\alpha}, \vec{R}) = 1 \ \wedge \ h_k(\vec{\alpha}, \vec{R}) = 1] = \Pr_{\vec{R}}[m_S(\vec{\alpha}, \vec{R}) = 0 \ \wedge \ h_k(\vec{\alpha}, \vec{R}) = 1] \ .$$

Since $\Pr_{\vec{R}}[h_k(\vec{\alpha}, \vec{R}) = 1] \neq 0$ by Claim 5, this implies the statement of the lemma. ◻

For $i = 1, \dots, q$ we denote by $\vec{\omega}_i(\vec{\alpha})$ the binary vector of length $2^{n\ell}$ that we get by replacing each 0 in $\vec{m}_i(\vec{\alpha})$ by 1, and replacing each 1 in $\vec{m}_i(\vec{\alpha})$ by $-1$. Similarly, for $\emptyset \neq S \subseteq [q]$ we denote by $\vec{\omega}_S(\vec{\alpha})$ the binary vector of length $2^{n\ell}$ that we get by replacing each 0 in $\vec{m}_S(\vec{\alpha})$ by 1, and replacing each 1 in $\vec{m}_S(\vec{\alpha})$ by $-1$. That is, we move from the domain $\{0, 1\}$ to the domain $\{1, -1\}$, and we obtain the vectors $\vec{\omega}_i(\vec{\alpha})$, for $i = 1, \dots, q$, from the vectors $\vec{m}_i(\vec{\alpha})$ using the standard transformation that replaces each value $b$ by $(-1)^b$. The vectors $\vec{\omega}_S(\vec{\alpha})$, for $\emptyset \neq S \subseteq [q]$, are obtained from the vectors $\vec{m}_S(\vec{\alpha})$ in the same way.

LEMMA 3.5. *Let $\vec{\omega}_1, \dots, \vec{\omega}_q$ be selected as above in a $t$-private protocol for $t \geq 2$. Then the vectors $\vec{\omega}_i(\vec{\alpha})$, $i = 1, \dots, q-1$, are linearly independent over the reals.*

*Proof.* As we will see in the next section, our job would be much easier (and we could obtain stronger bounds) if the vectors $\vec{h}_k(\vec{\alpha})$ were the same for each $k$. In that case we could show that the vectors $\vec{\omega}_i(\vec{\alpha})$, $i = 1, \dots, q-1$ (or some projections of them), are pairwise orthogonal. However, in general the vectors $\vec{h}_k(\vec{\alpha})$ may not be the same. Nevertheless, we can show that a given projection of each vector $\vec{\omega}_k(\vec{\alpha})$ is orthogonal to the same projection of each preceding vector $\vec{\omega}_i(\vec{\alpha})$ for $i < k$. This will let us show that for any $k$ such that $2 \leq k \leq q-1$, the vector $\vec{\omega}_k(\vec{\alpha})$ cannot be obtained as a linear combination of the vectors $\vec{\omega}_1(\vec{\alpha}), \dots, \vec{\omega}_{k-1}(\vec{\alpha})$.

Consider an arbitrary $k \in \{2, \dots, q-1\}$, and consider the following projection of the vectors $\vec{\omega}_1(\vec{\alpha}), \dots, \vec{\omega}_k(\vec{\alpha})$. Note that our choice of the projection depends on $k$ (via $\vec{h}_k(\vec{\alpha})$) and this is indicated in the notation by the superscript $k$. For $i = 1, \dots, k$, denote by $\vec{v}_i^k$ the projection of the vector $\vec{\omega}_i(\vec{\alpha})$ to only those coordinates where $h_k(\vec{\alpha}, \vec{R}) = 1$. Note that $\vec{h}_k(\vec{\alpha})$ is not identically 0, as stated in Claim 5, so there is always at least one such coordinate.

Since $t \geq 2$, by applying Lemma 3.4 to the sets $\{i, k\}$ for $i < k$ we get that the inner product of $\vec{v}_k^k$ with any of the vectors $\vec{v}_i^k$ for $i < k$ is 0. To see this, consider $\vec{\omega}_S(\vec{\alpha})$ for $S = \{i, k\}$. Considering $\{1, -1\}$ vectors instead of $\{0, 1\}$ vectors, Lemma 3.4 states that

$$\Pr_{\vec{R}}[\omega_S(\vec{\alpha}, \vec{R}) = 1 | h_k(\vec{\alpha}, \vec{R}) = 1] = \Pr_{\vec{R}}[\omega_S(\vec{\alpha}, \vec{R}) = -1 | h_k(\vec{\alpha}, \vec{R}) = 1] = 1/2 \ .$$

Thus, $\sum_{\{\vec{R}:h_k(\vec{\alpha},\vec{R})=1\}} \omega_S(\vec{\alpha},\vec{R}) = 0$. Notice that for $S = \{i,k\}$, the above sum is exactly the inner product of the vectors $\vec{v}_i^k$ and $\vec{v}_k^k$, since $\omega_S(\vec{\alpha},\vec{R}) = \omega_i(\vec{\alpha},\vec{R}) \cdot \omega_k(\vec{\alpha},\vec{R})$. Therefore, we get that the inner product of $\vec{v}_k^k$ with any of the vectors $\vec{v}_i^k$ for $i < k$ is 0, as claimed.

Suppose that $\vec{\omega}_k(\vec{\alpha})$ can be obtained as a linear combination of the vectors $\vec{\omega}_1(\vec{\alpha}),\ldots,\vec{\omega}_{k-1}(\vec{\alpha})$. Then $\vec{v}_k^k$ can be obtained as a linear combination of the vectors $\vec{v}_1^k,\ldots,\vec{v}_{k-1}^k$. But since the inner product of $\vec{v}_k^k$ with each $\vec{v}_i^k$ for $i < k$ is 0, this would imply that the inner product of $\vec{v}_k^k$ with itself is 0. Since $\vec{v}_k^k$ has only 1 or $-1$ entries, this is not possible. ▯

Let us now consider the $(q-1) \times 2^{n\ell}$ matrix, formed by the vectors $\vec{\omega}_i(\vec{\alpha})$, $i = 1,\ldots,q-1$, as row vectors. Since the vectors $\vec{\omega}_i(\vec{\alpha})$, $i = 1,\ldots,q-1$, are linearly independent, this matrix has at least $q-1$ different columns. This implies that the protocol has at least $q-1$ different transcripts on input $\vec{\alpha}$. Since $q = \Omega(n)$, Theorem 1.1 follows by Lemma 2.6.

## 4. Restricted protocols.

In this section we consider a class of restricted protocols that we define below. Our motivation to consider this class is that all known randomness-efficient protocols designed specifically for `xor` obey this restriction, or can be easily brought to this form without changing the number of coin tosses performed [27, 23, 26, 8]. Informally one can describe the protocols of this class in the following way. First, a deterministic, nonprivate protocol to compute $f$ is defined. Then this protocol is modified by masking each message with randomness by adding to it (modulo 2) a value that depends on the randomness only. This restriction was previously considered in [27]. Formally the restriction we consider here is defined as follows.

DEFINITION 4.1. *We say that a given protocol involving* $n$ *players has a restricted form if each message of the protocol can be obtained as a mod 2 sum of a Boolean function* $u : \{0,1\}^n \to \{0,1\}$ *that depends on the input variables only, and a Boolean function* $v : \{0,1\}^{n\ell} \to \{0,1\}$ *that depends on the random tapes only. That is, each message* $m$ *can be written as* $m(\vec{x},\vec{R}) = u(\vec{x}) + v(\vec{R})$ *mod* 2.

THEOREM 4.2. *Let* $t \geq 2$*, and let* $\mathcal{A}$ *be a* $d$*-random,* $t$*-private protocol, obeying the above restriction, for computing* $f(\vec{x}) = x_1 + \cdots + x_n$ *(mod 2). Then* $d = \Omega(t\log(n/t))$.

As in our proof for general protocols, here too we prove in fact a stronger claim. We prove that for any input assignment $\vec{\alpha} = \alpha_1,\ldots,\alpha_n$, there is a run of the protocol in which the number of random bits read by the players from their random tapes is $\Omega(t\log(n/t))$.

*Proof.* Let $\vec{\alpha} = \alpha_1,\ldots,\alpha_n$ be an arbitrary input assignment. We select a sequence of messages $m_1,\ldots,m_q$ and define the corresponding vectors as in the previous section. Recall that $\vec{h}_i(\vec{\alpha})$ denotes the bitwise mod 2 sum of the vectors $\vec{m}_i(\vec{\alpha})$ and $\vec{m}_i(\vec{\alpha}^{(i)})$. The restriction on the protocols we consider allows us to have the following claim.

CLAIM 6. *The vectors* $\vec{h}_i(\vec{\alpha})$*,* $i = 1,\ldots,q$*, are identically* 1.

*Proof.* We know by Claim 5 that the vectors $\vec{h}_i(\vec{\alpha})$ are not identically 0. That is, they have at least one entry with value 1. This means that $m_i(\vec{\alpha},\vec{R}) \neq m_i(\vec{\alpha}^{(i)},\vec{R})$ for at least one $\vec{R}$. But $m_i(\vec{x},\vec{R})$ can be written as $m_i(\vec{x},\vec{R}) = u_i(\vec{x}) + v_i(\vec{R})$ mod 2. It follows that $u_i(\vec{\alpha}) \neq u_i(\vec{\alpha}^{(i)})$, and therefore for any $\vec{R}$, $m_i(\vec{\alpha},\vec{R}) \neq m_i(\vec{\alpha}^{(i)},\vec{R})$. ▯

The above claim allows us to obtain a stronger bound using the machinery of the previous section. We now prove the following lemma.

LEMMA 4.3. *Let* $\vec{\omega}_S$ *be defined as above in a* $t$*-private protocol obeying the above* *restriction for $t \geq 2$. Then the vectors $\vec{\omega}_S(\vec{\alpha})$, such that $\emptyset \neq S \subseteq [q-1]$ and $|S| \leq \lfloor t/2 \rfloor$, are linearly independent over the reals.*

*Proof.* First note that since $t \geq 2$, there exist sets $S$, such that $\emptyset \neq S \subseteq [q-1]$ and $|S| \leq \lfloor t/2 \rfloor$; thus the statement of the lemma is meaningful (assuming $n \geq 5$; otherwise $q-1$ could be less than 1, since the guarantee of Lemma 3.2 is that $q \geq \frac{n}{4}$). Since by Claim 6 for each $i \in [q]$ the vector $\vec{h}_i(\vec{\alpha})$ is identically 1, Lemma 3.4 gives that $\Pr_{\vec{R}}[m_T(\vec{\alpha}, \vec{R}) = 1] = 1/2$ for any $\emptyset \neq T \subseteq [q-1]$ of size at most $t$. This implies that the sum of entries of the vector $\vec{\omega}_T(\vec{\alpha})$ is 0 for any $\emptyset \neq T \subseteq [q-1]$ of size at most $t$. Notice that for any two sets $S_1$ and $S_2$, we have $\omega_{S_1}(\vec{\alpha}, \vec{R}) \cdot \omega_{S_2}(\vec{\alpha}, \vec{R}) = \omega_{S_1 \triangle S_2}(\vec{\alpha}, \vec{R})$. Thus, for sets $\emptyset \neq S_1 \subseteq [q-1]$ and $\emptyset \neq S_2 \subseteq [q-1]$, each of size at most $\lfloor t/2 \rfloor$, the inner product of $\vec{\omega}_{S_1}(\vec{\alpha})$ and $\vec{\omega}_{S_2}(\vec{\alpha})$ must be 0. We get that the vectors $\vec{\omega}_S(\vec{\alpha})$ for $\emptyset \neq S \subseteq [q-1]$ and $|S| \leq \lfloor t/2 \rfloor$ are pairwise orthogonal, and therefore they must be linearly independent over the reals. ∎

We denote by $\binom{a}{\leq b}$ the sum $\sum_{i=1}^{\min(a,b)} \binom{a}{i}$ for integers $a, b \geq 1$. Let us now consider the $\binom{q-1}{\leq \lfloor t/2 \rfloor} \times 2^{n\ell}$ matrix, formed by the vectors $\vec{\omega}_S(\vec{\alpha})$, such that $\emptyset \neq S \subseteq [q-1]$ and $|S| \leq \lfloor t/2 \rfloor$, as row vectors. Since the vectors $\vec{\omega}_S(\vec{\alpha})$ are linearly independent, this matrix has at least $\binom{q-1}{\leq \lfloor t/2 \rfloor}$ different columns. Note that each column of the matrix is associated with a fixed vector of random tapes $\vec{R}$, and each column is completely determined by the transcript of the protocol on the given $\vec{R}$ and $\vec{\alpha}$. This implies that the protocol has at least $\binom{q-1}{\leq \lfloor t/2 \rfloor}$ different transcripts on input $\vec{\alpha}$. Since $q = \Omega(n)$, the theorem follows by Lemma 2.6. ∎

We find it worthwhile to note that our proof also implies that the random variables associated with the messages $m_1, \ldots, m_q$ on input $\vec{\alpha}$ are $t$-wise independent in any $t$-private protocol that obeys the above restriction. Thus, we could conclude the proof of Theorem 4.2 by referring to the known lower bounds on the size of sample spaces with $t$-wise independent random variables [12, 1]. In fact, part of the proof of Lemma 4.3 is analogous to the corresponding part of the argument used in [12]. The $t$-wise independence property of the random variables associated with the messages $m_1, \ldots, m_q$ on input $\vec{\alpha}$ follows by Lemma 3.4, which, as we have shown above, gives in the case of the restricted protocols that $\Pr_{\vec{R}}[m_S(\vec{\alpha}, \vec{R}) = 1] = 1/2$ for any $\emptyset \neq S \subseteq [q-1]$ of size at most $t$. This implies $t$-wise independence of the corresponding random variables by the results of [12].

## REFERENCES

[1] N. Alon and J. H. Spencer, *The Probabilistic Method*, 2nd ed., John Wiley, New York, 2000.

[2] J. Bar-Ilan and D. Beaver, *Non-cryptographic fault-tolerant computing in a constant number of rounds*, in Proceedings of the 8th Symposium on Principles of Distributed Computing (PODC), 1989, pp. 201–209.

[3] D. Beaver, *Perfect privacy for two-party protocols*, in Distributed Computing and Cryptography, DIMACS Ser. Discrete Math. Theoret. Comput. Sci. 2, AMS, Providence, RI, 1991, pp. 65–77.

[4] M. Ben-or, S. Goldwasser, and A. Wigderson, *Completeness theorems for non-cryptographic fault-tolerant distributed computation*, in Proceedings of the 20th Symposium on Theory of Computing (STOC), 1988, pp. 1–10.

[5] M. Bläser, A. Jakoby, M. Liskiewicz, and B. Siebert, *Private computation—k-connected versus* 1*-connected networks*, in Proceedings of the 22nd CRYPTO, 2002, pp. 194–209.

[6] C. Blundo, A. De Santis, G. Persiano, and U. Vaccaro, *On the number of random bits in totally private computations*, in Proceedings of the 22nd International Conference on Automata, Languages and Programming (ICALP), 1995, pp. 171–182.

[7] C. Blundo, A. De Santis, G. Persiano, and U. Vaccaro, *Randomness complexity of private computation*, Comput. Complexity, 8 (1999), pp. 145–168.

[8] C. Blundo, C. Galdi, and P. Persiano, *Randomness recycling in constant-round private computations*, in Proceedings of the 13th International Symposium on Distributed Computing (DISC), 1999, pp. 138–150.

[9] C. Blundo, C. Galdi, and P. Persiano, *Low-Randomness Constant-Round Private Computations*, manuscript, 2003.

[10] R. Canetti, E. Kushilevitz, R. Ostrovsky, and A. Rosén, *Randomness vs. fault-tolerance*, J. Cryptology, 13 (2000), pp. 107–142.

[11] D. Chaum, C. Crepeau, and I. Damgard, *Multiparty unconditionally secure protocols*, in Proceedings of the 20th Symposium on Theory of Computing (STOC), 1988, pp. 11–19.

[12] B. Chor, J. Friedmann, O. Goldreich, J. Hastad, S. Rudich, and R. Smolensky, *The bit extraction problem and t-resilient functions*, in Proceedings of the 26th IEEE Symposium on Foundations of Computer Science (FOCS), 1985, pp. 396–407.

[13] B. Chor and E. Kushilevitz, *A zero-one law for Boolean privacy*, SIAM J. Discrete Math., 4 (1991), pp. 36–47.

[14] B. Chor, M. Geréb-Graus, and E. Kushilevitz, *Private computations over the integers*, SIAM J. Comput., 24 (1995), pp. 376–386.

[15] U. Feige, J. Kilian, and M. Naor, *A minimal model for secure computation*, in Proceedings of the 26th Symposium on Theory of Computing (STOC), 1994, pp. 554–563.

[16] M. Franklin and M. Yung, *Communication complexity of secure computation*, in Proceedings of the 24th Symposium on Theory of Computing (STOC), 1992, pp. 699–710.

[17] A. Gál and A. Rosén, *A theorem on sensitivity and applications in private computation*, SIAM J. Comput., 31 (2002), pp. 1424–1437.

[18] O. Goldreich, *Modern Cryptography, Probabilistic Proofs and Pseudorandomness*, Algorithms Combin. 17, Springer-Verlag, Berlin, 1998.

[19] O. Goldreich, S. Micali, and A. Wigderson, *How to play any mental game*, in Proceedings of the 19th Symposium on Theory of Computing (STOC), 1987, pp. 218–229.

[20] A. Jakoby, M. Liskiewicz, and R. Reischuk, *Private computations in networks: Topology vs. randomness*, in Proceedings of the 21st Annual Symposium on Theoretical Aspects of Computer Science (STACS), 2003, pp. 121–132.

[21] J. Kilian, E. Kushilevitz, S. Micali, and R. Ostrovsky, *Reducibility and completeness in private computations*, SIAM J. Comput., 29 (2000), pp. 1189–1208.

[22] S. C. Kleene, *Mathematical Logic*, Dover, New York, 2002.

[23] E. Kushilevitz and Y. Mansour, *Randomness in private computations*, SIAM J. Discrete Math., 10 (1997), pp. 647–661.

[24] E. Kushilevitz, *Privacy and communication complexity*, SIAM J. Discrete Math., 5 (1992), pp. 273–284.

[25] E. Kushilevitz, R. Ostrovsky, and A. Rosén, *Characterizing linear size circuits in terms of privacy*, J. Comput. System Sci., 58 (1999), pp. 129–136.

[26] E. Kushilevitz, R. Ostrovsky, and A. Rosén, *Amortizing randomness in private multiparty computations*, SIAM J. Discrete Math., 16 (2003), pp. 533–544.

[27] E. Kushilevitz and A. Rosén, *A randomness-rounds tradeoff in private computation*, SIAM J. Discrete Math., 11 (1998), pp. 61–80.

[28] N. Nisan and A. Ta-Shma, *Extracting randomness: A survey and new constructions*, J. Comput. System Sci., 58 (1999), pp. 148–173.

[29] A. Yao, *Protocols for secure computation*, in Proceedings of the 23rd IEEE Symposium on Foundations of Computer Science (FOCS), 1982, pp. 160–164.

SIAM J. COMPUT.
Vol. 34, No. 4, pp. 960–992

# A LOWER BOUND ON THE COMPLEXITY OF POLYNOMIAL MULTIPLICATION OVER FINITE FIELDS*

MICHAEL KAMINSKI†

**Abstract.** It is shown that computing the coefficients of the product of two degree-$n$ polynomials over a $q$-element field by means of a quadratic algorithm requires at least $(3 + \frac{(q-1)^2}{q^5+(q-1)^3})n - o(n)$ multiplications.

**Key words.** polynomial multiplication, quadratic algorithms, linear recurring sequences, Hankel matrices, error-correcting codes

**AMS subject classifications.** 11B37, 11C08, 11T06, 11T71, 12Y05

**DOI.** 10.1137/S0097539704442118

## 1. Introduction.

In infinite fields it is possible to compute the coefficients of the product of two polynomials of degree $n$ in $2n + 1$ nonscalar multiplications. It is known from [23] that each algorithm for computing the product in $2n + 1$ nonscalar multiplications must evaluate the multiplicands at $2n + 1$ distinct points (possibly including $\infty$), multiply the samples, and interpolate the result. However, in finite fields this method fails if $2n$ exceeds the number of the field elements. Thus, in general, the $2n + 1$ tight bound cannot be achieved in finite fields.

Let $\mu_F(n)$ denote the number of multiplications required to compute the coefficients of the product of a polynomial of degree $n$ and a polynomial of degree $n - 1$ over field $F$ by means of quadratic algorithms¹ and let $F_q$ denote the $q$-element field. For $q \geq 3$, the best lower bound on $\mu_{F_q}(n)$ known from the literature is $3n - o(n)$ (see [15] and [6, Theorem 18.10]) and, for sufficiently large $n$, $\mu_{F_2}(n) > 3.52n$ (see [3]). On the other hand, if $q \geq 3$, computing the coefficients of the minimal degree residue of the product of two polynomials of degree $n$ modulo a fixed irreducible polynomial of degree $n + 1$ over $F_{q^2}$ can be done in $2(1 + \frac{1}{q-2})n + o(n)$ multiplications; see [7] and [20]. Thus, for $q \geq 3$, $\mu_{F_{q^2}}(n) \leq 4(1+\frac{1}{q-2})n+o(n)$. This small difference between the upper and the lower bounds motivates a further search for better (both upper and lower) bounds on the complexity of polynomial multiplications. Also, apart from being of interest in their own right, algorithms for polynomial multiplication over finite fields are tightly connected to error-correcting codes; see [2, 3, 7, 14, 16, 17].

In our paper we prove the following lower bound on $\mu_{F_q}(n)$.

THEOREM 1. *We have*

$$\mu_{F_q}(n) \geq \left(3 + \frac{(q-1)^2}{q^5 + (q-1)^3}\right) n - o(n).$$

*Received by the editors March 18, 2004; accepted for publication (in revised form) November 16, 2004; published electronically May 12, 2005. The results of this paper were presented at the 22nd Symposium on Theoretical Aspects of Computer Science (STACS 2005), Stuttgart, Germany. This research was supported by the Technion V.P.R. fund.
http://www.siam.org/journals/sicomp/34-4/44211.html

†Department of Computer Science, Technion – Israel Institute of Technology, Haifa 32000, Israel (kaminski@cs.technion.ac.il).

¹A straightforward substitution argument shows that the number of multiplications required to compute the coefficient of the product of two polynomials of degree $n$ over $F$ exceeds $\mu_F(n)$ by at least 1.

The proof of Theorem 1 is based on a novel combination of two known techniques. One technique is the analysis of *Hankel* matrices representing bilinear forms defined by linear combinations of the coefficients of the polynomial product; see [15]. The other technique is a counting argument from the coding theory; see [16].

The reason for combining these two techniques is that the Hankel matrix approach uses very few properties of finite fields and the coding approach does not use at all a very special structure of bilinear forms defined by linear combinations of the coefficients of the polynomial product. In fact, our paper indicates that the Hankel matrix approach is, in some sense, richer than Baur's technique; see [6, proof of Theorem 18.10]. In particular, as a byproduct, using the tools developed for the proof of Theorem 1, we obtain a more intuitive alternative proof of the lower bound on the complexity of multiplication of polynomials modulo a polynomial established in [4] and [7].

We end this section with the note that if a set of quadratic forms over an infinite field can be computed by a (general) *straight-line algorithm* in $t$ multiplications/divisions, then it can be computed in $t$ multiplications by a quadratic algorithm whose total number of operations differs from that of the original one by a factor of a small constant; see [21]. It is unknown whether a similar result holds for finite fields; cf. [5]. However, the proof in [21] easily extends to finite fields if, for some input that belongs to the underlying finite field, the original straight-line algorithm does not divide by zero. Therefore, in the case of multiplication of polynomials over $F_q$, our lower bound applies to all straight-line algorithms which compute the coefficients of the product of at least one pair of polynomials whose coefficients all belong to $F_q$. Finally, one can easily prove that quadratic algorithms for computing a set of bilinear forms are optimal within the class of algorithms without divisions, and all algorithms for polynomial multiplication over finite fields known from the literature are quadratic (and even *bilinear*).

The paper is organized as follows. In the next section we gather the definitions and some basic facts used in this paper. In section 3 we indicate some limitations of the known tools and present an example through which we develop a technique used for the proof of Theorem 1. Then, in section 4, we outline the proof of the theorem. The proof itself is based on an improvement of a known lower bound presented in section 5 and a number of technical constructions presented in sections 6, 7, and 8. In particular, the improved lower bound in section 5 is based on a counting argument from the coding theory and can be used for an alternative proof of the complexity of multiplication of polynomials modulo a polynomial.

## 2. Notation, definitions, and auxiliary results.

This section contains the definitions and some basic results known from the literature which we use in this paper. First we define the notion of a quadratic algorithm. Then we introduce notation and definitions from linear recurring sequence theory and state the major auxiliary technical lemmas. We conclude this section with some basic calculations from the coding theory and an example that will be used for the proof of Theorem 1.

### 2.1. Quadratic algorithms for polynomial multiplication.

In this paper we deal only with quadratic algorithms defined below.

Let $\boldsymbol{s}$ be a set of indeterminates. We remind the reader that a *quadratic* algorithm over field $F$ for computing a set $\boldsymbol{Q}$ of quadratic forms of the elements of $\boldsymbol{s}$ is a straight-line algorithm whose nonscalar multiplications are of the form $L' {*} L''$, where $L'$ and $L''$ are linear forms of the elements of $\boldsymbol{s}$ over $F$ and each form in $\boldsymbol{Q}$ is a linear combination of these products. The minimal number of such multiplications is called the *quadratic complexity of $\boldsymbol{Q}$ over $F$* and is denoted by $\mu_F(\boldsymbol{Q})$.

Let $\boldsymbol{Q} = \{\boldsymbol{x}^T M \boldsymbol{y} : M \in \boldsymbol{M}\}$, where $\boldsymbol{M}$ is a set of matrices. That is, the quadratic (in fact, *bilinear*) forms in $\boldsymbol{Q}$ are defined by the elements of $\boldsymbol{M}$.

*In what follows we shall identify $\boldsymbol{Q}$ with $\boldsymbol{M}$ and often write $\mu_F(\boldsymbol{M})$ instead of $\mu_F(\boldsymbol{Q})$. Also, we shall identify a quadratic algorithm with the corresponding set of pairs of linear forms.*

Let $\boldsymbol{x} = (x_0, x_1, \ldots, x_n)^T$ and $\boldsymbol{y} = (y_0, y_1, \ldots, y_{n-1})^T$ be column vectors of the multiplicands' coefficients. We have to compute

$$z_k = z_k(\boldsymbol{x}, \boldsymbol{y}) = \sum_{i+j=k} x_i y_j, \quad k = 0, \ldots, 2n-1. \tag{1}$$

Assume that $\mu_F(n) = t$, i.e., there exist $t$ linear forms $L_1'(\boldsymbol{x}, \boldsymbol{y}), \ldots, L_t'(\boldsymbol{x}, \boldsymbol{y})$ and $t$ linear forms $L_1''(\boldsymbol{x}, \boldsymbol{y}), \ldots, L_t''(\boldsymbol{x}, \boldsymbol{y})$ of $\boldsymbol{x}$ and $\boldsymbol{y}$ such that each $z_k$ is a linear combination of products $L_i'(\boldsymbol{x}, \boldsymbol{y}) L_i''(\boldsymbol{x}, \boldsymbol{y})$, $i = 1, 2, \ldots, t$.

Let $\boldsymbol{z} = (z_0, z_1, \ldots, z_{2n-1})^T$ and let $\boldsymbol{p}$ be a column vector of the above products. That is,

$$\boldsymbol{p} = \begin{pmatrix} L_1'(\boldsymbol{x}, \boldsymbol{y}) L_1''(\boldsymbol{x}, \boldsymbol{y}) \\ L_2'(\boldsymbol{x}, \boldsymbol{y}) L_2''(\boldsymbol{x}, \boldsymbol{y}) \\ \vdots \\ L_t'(\boldsymbol{x}, \boldsymbol{y}) L_t''(\boldsymbol{x}, \boldsymbol{y}) \end{pmatrix},$$

say. Then there exists a $2n \times t$ matrix $U$ such that $\boldsymbol{z} = U\boldsymbol{p}$.

By definition, $z_k = \boldsymbol{x}^T A_k \boldsymbol{y}$, where $A_k = (a_{i,j,k})$ is an $(n+1) \times n$ Hankel matrix defined by

$$a_{i,j,k} = \begin{cases} 1 & \text{if } i + j = k + 2, \\ 0 & \text{otherwise.} \end{cases}$$

In particular, $z_0 = \boldsymbol{x}^T A_0 \boldsymbol{y} = x_0 y_0$ is the value of polynomial $\sum_{k=0}^{2n-1} z_k \alpha^k$ at zero, and $z_{2n-1} = \boldsymbol{x}^T A_{2n-1} \boldsymbol{y} = x_n y_{n-1}$ is its value at infinity.

Since matrices $A_0, A_1, \ldots, A_{2n-1}$ are linearly independent, $\operatorname{rank} U = 2n$. Permuting the components of $\boldsymbol{p}$ if necessary, we may assume that the first $2n$ columns of $U$ are linearly independent. Hence there exist a nonsingular $2n \times 2n$ matrix $W$ and a $2n \times (t - 2n)$ matrix $V$ such that

$$W\boldsymbol{z} = (\boldsymbol{I}_{2n}, V)\boldsymbol{p}, \tag{2}$$

where $\boldsymbol{I}_{2n}$ denotes the $2n \times 2n$ identity matrix. That is, the first $2n$ columns of $WU$ are those of $\boldsymbol{I}_{2n}$.

Let $W\boldsymbol{z} = (\boldsymbol{x}^T D_1 \boldsymbol{y}, \boldsymbol{x}^T D_2 \boldsymbol{y}, \ldots, \boldsymbol{x}^T D_{2n} \boldsymbol{y})^T$ and $\boldsymbol{D} = \{D_1, D_2, \ldots, D_{2n}\}$. Then $\mu_F(n) = \mu_F(\boldsymbol{D})$, which together with (2) easily implies Proposition 2 below.

PROPOSITION 2 (cf. [1, Proposition 1]). *Let $\boldsymbol{D}$ be as above and let $\boldsymbol{D}' \subseteq \boldsymbol{D}$. Then*

$$\mu_F(n) \geq 2n + 1 + \mu_F(\boldsymbol{D}') - |\boldsymbol{D}'|.[2]$$

We apply Proposition 2 for the proof of Theorem 1 in a standard manner; see [1], [5], [6, proof of Theorem 18.10], and [15]. Namely, we prove that there exists a subset $\boldsymbol{D}'$ of $\boldsymbol{D}$ of cardinality $o(n)$ such that

$$\mu_{F_q}(\boldsymbol{D}') \geq \left(1 + \frac{(q-1)^2}{q^5 + (q-1)^3}\right) n;$$

[2]As usual, for a set $X$, $|X|$ denotes the number of elements of $X$.

cf. [6, proof of Theorem 18.10] and [15]. For this purpose we refine the technique developed in [15].

**2.2. Hankel matrices and linear recurring sequences.** In this section we recall the notation and state major auxiliary lemmas from [5] and [15].

Let $F$ be a field, let $k$ be a positive integer, and let $a_0, \ldots, a_{k-1} \in F$. A sequence $\sigma = s_0, s_1, \ldots, s_\ell$ of elements of $F$ satisfying the relation

$$s_{m+k} = a_{k-1}s_{m+k-1} + a_{k-2}s_{m+k-2} + \cdots + a_0 s_m, \quad m = 0, 1, \ldots, \ell - k,$$

is called a (finite $k$th-order homogeneous) *linear recurring sequence* in $F$. The terms $s_0, s_1, \ldots, s_{k-1}$ are called *initial values*, and the polynomial

$$f(\alpha) = \alpha^k - a_{k-1}\alpha^{k-1} - a_{k-2}\alpha^{k-2} - \cdots - a_0 \in F[\alpha]$$

is called a *characteristic polynomial* of $\sigma$. The characteristic polynomial of $\sigma$ of the minimal degree is called the *minimal polynomial* of $\sigma$ and is denoted by $f_\sigma(\alpha)$.

Proposition 3 below shows that if a finite linear recurring sequence is "sufficiently long," then it possesses an important property of infinite linear recurring sequences.

PROPOSITION 3 (see [15, Proposition 1]). *Let $\sigma$, $f(\alpha)$, and $f_\sigma(\alpha)$ be as above. If* $\deg f_\sigma(\alpha) + \deg f(\alpha) \le \ell + 1$, *then $f_\sigma(\alpha)$ divides $f(\alpha)$.*

For a sequence $\sigma = s_0, s_1, \ldots, s_{2n-1}$ we define the $(n+1)\times n$ *Hankel* matrix $H(\sigma)$ by

$$H(\sigma) = \begin{pmatrix} s_0 & s_1 & \cdots & s_{n-1} \\ s_1 & s_2 & \cdots & s_n \\ \vdots & \vdots & & \vdots \\ s_n & s_{n+1} & \cdots & s_{2n-1} \end{pmatrix}.$$

The proof of Theorem 1 is based on the fact that linear combinations of the coefficients of the product of two polynomials are defined by Hankel matrices and vice versa.

We proceed with Definition 4 below, for which we need the following notation.

We denote by $\vec{H}^i$, $i = 0, 1, \ldots, n$, the $(i+1)$th row of a $(n+1)\times n$ Hankel matrix $H$.

DEFINITION 4. *Let $\sigma = s_0, s_1, \ldots, s_{2n-1}$, $H = H(\sigma)$, and let $k$ be the minimal integer for which there exist $a_0, a_1, \ldots, a_{k-1} \in F$ such that*

$$\sum_{i=0}^{k-1} a_i \vec{H}^i = \vec{H}^k.$$

*The sequence $\tilde{\sigma} = \tilde{s}_0, \tilde{s}_1, \ldots, \tilde{s}_{2n-1}$ is defined by linear recurrence*

$$\tilde{s}_{i+k} = a_{k-1}\tilde{s}_{i+k-1} + a_{k-2}\tilde{s}_{i+k-2} + \cdots + a_0\tilde{s}_i, \tag{3}$$

*with initial values $\tilde{s}_i = s_i$, $i = 0, 1, \ldots, k-1$.*

*The sequence $\sigma - \tilde{\sigma}$ is denoted by $\bar{\sigma}$, and $(n+1)\times n$ Hankel matrices $H(\tilde{\sigma})$ and $H(\bar{\sigma})$ are denoted by $\tilde{H}$ and $\bar{H}$, respectively.*

*We denote characteristic polynomial $\alpha^k - \sum_{i=0}^{k-1} a_i\alpha^i$ of the sequence defined by recurrence* (3) *by $f_{\tilde{H}}(\alpha)$*[3] *and we define the (integral)* divisor $\boldsymbol{f}_H(\alpha)$ *by*

$$\boldsymbol{f}_H(\alpha) = f_{\tilde{H}}(\alpha)(\alpha - \infty)^{rank(\bar{H})}.$$

[3]In fact, $f_{\tilde{H}}(\alpha)$ is the *minimal* polynomial of this sequence and, in particular, of $\tilde{\sigma}$; see [15, section 3].

*Example* 5. Let $d \leq n-1$ and let $\sigma = s_0, s_1, \ldots, s_{2n}$ be such that $s_d \neq 0$ and $s_j = 0$ for all $j = d+1, d+2, \ldots, 2n$. Then $\boldsymbol{f}_{H(\sigma)}(\alpha) = \alpha^d$. Similarly, for $\sigma' = s'_0, s'_1, \ldots, s'_{2n}$, where $s'_{2n-d} \neq 0$ and $s'_j = 0$ for all $j = 0, 1, \ldots, 2n-d-1$, $\boldsymbol{f}_{H(\sigma')}(\alpha) = (\alpha - \infty)^d$. Thus, in particular, for the zero $(n+1) \times n$ matrix $\mathbf{0}_{(n+1)\times n}$, $\boldsymbol{f}_{\mathbf{0}_{(n+1)\times n}}(\alpha) = 1$.

PROPOSITION 6 (see [15, Proposition 2]). *Let $H$ be an $(n+1)\times n$ Hankel matrix. Then* $\operatorname{rank}(H) = deg \boldsymbol{f}_H(\alpha)$.

A part of the proof of Theorem 1 is based on Lemmas 7–11 below. To state these lemmas we need the following notation.

Let $\boldsymbol{S}$ be a set of $(n+1) \times n$ Hankel matrices. We denote by $\boldsymbol{f}_{\boldsymbol{S}}(\alpha)$ the least common multiple of $\{\boldsymbol{f}_H(\alpha) : H \in \boldsymbol{S}\}$:

$$\boldsymbol{f}_{\boldsymbol{S}}(\alpha) = \operatorname{lcm}\{\boldsymbol{f}_H(\alpha) : H \in \boldsymbol{S}\}.$$

Let $V$ be a vector space over $F$ and let $V' \subseteq V$. We denote by $[V']$ the linear subspace of $V$ spanned by the elements of $V'$.

Finally, we denote the maximal possible number of distinct factors of a polynomial of degree $n$ over $F_q$ by $i_q(n)$. It is well known that for $q \geq 3$, $i_q(n) < \frac{n}{\lg_q n - 3}$; e.g., see [15, Appendix 1].

LEMMA 7 (see [15, Lemma 1]). *Let $\boldsymbol{S}$ be a set of $(n+1)\times n$ Hankel matrices. Then* $\dim([\boldsymbol{S}]) \leq \deg \boldsymbol{f}_{\boldsymbol{S}}(\alpha)$.

LEMMA 8 (see [15, Lemma 2] and [5, Lemma 1]). *Let $\boldsymbol{S}$ be a set of $(n+1)\times n$ Hankel matrices over a field $F$ such that* $\deg \boldsymbol{f}_{\boldsymbol{S}}(\alpha) \geq n+1$. *Then* $\mu_F(\boldsymbol{S}) \geq n+1$.

LEMMA 9 (see [15, Lemma 3] and [5, Lemma 1]). *Let $\boldsymbol{S}$ be a set of $(n+1)\times n$ Hankel matrices over a field $F$ such that* $\deg \boldsymbol{f}_{\boldsymbol{S}}(\alpha) < n+1$. *Then* $\mu_F(\boldsymbol{S}) \geq \deg \boldsymbol{f}_{\boldsymbol{S}}(\alpha)$.

LEMMA 10 (see [5, Lemma 2]). *Let $\boldsymbol{S}$ and $\boldsymbol{S}'$ be sets of $(n+1)\times n$ Hankel matrices such that $[\boldsymbol{S}] = [\boldsymbol{S}']$. If* $\deg \boldsymbol{f}_{\boldsymbol{S}}(\alpha) \leq n$, *then* $\boldsymbol{f}_{\boldsymbol{S}'}(\alpha) = \boldsymbol{f}_{\boldsymbol{S}}(\alpha)$.

LEMMA 11 (see [15, Lemma 4]). *Let $\boldsymbol{M}$ be a set of $(n+1)\times n$ Hankel matrices over $F_q$. Then for each $m \leq \dim(\boldsymbol{M})$ there exists a subset $\boldsymbol{M}'$ of $\boldsymbol{M}$ containing $i_q(m)$ or fewer elements such that* $\deg \boldsymbol{f}_{\boldsymbol{M}'}(\alpha) \geq m$.

The following example illustrates the power of the tools listed so far.

*Example* 12 (see [15, proof of Theorem 1]). Let $\boldsymbol{D}$ be the set of Hankel matrices defining the components of $W\boldsymbol{z}$ in (2). Then $\dim(\boldsymbol{D}) = 2n$ and, by Lemma 11, there is a subset $\boldsymbol{D}'$ of $\boldsymbol{D}$ of cardinality not exceeding $i_q(n+1)$ such that $\deg \boldsymbol{f}_{\boldsymbol{D}'}(\alpha) \geq n+1$. By Lemma 8, $\mu_{F_q}(\boldsymbol{D}') \geq n+1$ which, together with Proposition 2, implies $\mu_{F_q}(n) = 3n - o(n)$.

**2.3. Bounds from the coding theory.** This section contains some elementary calculations from the coding theory needed for the proof of Theorem 1.

The most basic notion of the coding theory is the *weight* of a vector $\boldsymbol{v}$, denoted $\boldsymbol{wt}(\boldsymbol{v})$, that is the number of nonzero components of $\boldsymbol{v}$.

PROPOSITION 13 (see [19, problem 3.6, p. 73]). *Let $U$ be a $k \times t$ matrix over $F_q$ without zero columns. Then*

$$\sum_{\boldsymbol{v} \in F_q^k} \boldsymbol{wt}(\boldsymbol{v}U) = t(q^k - q^{k-1}).$$

COROLLARY 14. *Let $U$ be a $2 \times t$ matrix over $F_q$ without zero columns. Then*

$$t = \frac{\boldsymbol{wt}((0,1)U) + \sum_{v \in F_q} \boldsymbol{wt}((1,v)U)}{q}.$$

The proof of this corollary is immediate and is left to the reader.

COROLLARY 15 (cf. [16, proof of Theorem 1]). *Let $\boldsymbol{S} = \{H_1, H_2, \ldots, H_k\}$ be a set of $(n+1) \times n$ Hankel matrices over $F_q$. Then*

$$\mu_{F_q}(\boldsymbol{S}) \geq \frac{\sum_{H \in [\boldsymbol{S}]} \deg \boldsymbol{f}_H(\alpha)}{q^k - q^{k-1}}.$$

*Proof.* Let $\mu_{F_q}(\boldsymbol{S}) = t$ and let $\{(L_i'(\boldsymbol{x},\boldsymbol{y}), L_i''(\boldsymbol{x},\boldsymbol{y}))\}_{i=1,2,\ldots,t}$ be a quadratic algorithm over $F_q$ that computes the bilinear forms defined by the elements of $\boldsymbol{S}$. Let $\boldsymbol{p}$ be a column vector of products $L_i'(\boldsymbol{x},\boldsymbol{y})L_i''(\boldsymbol{x},\boldsymbol{y})$, $i = 1, 2, \ldots, t$, and let $U$ be a $k \times t$ matrix over $F_q$ such that

$$(\boldsymbol{x}^T H_1 \boldsymbol{y}, \boldsymbol{x}^T H_2 \boldsymbol{y}, \ldots, \boldsymbol{x}^T H_k \boldsymbol{y})^T = U\boldsymbol{p}.$$

Let $\vec{\boldsymbol{S}} = (H_1, H_2, \ldots, H_k)^T$. Then, for each $\boldsymbol{v} \in F_q^k$,

$$\boldsymbol{x}^T(\boldsymbol{v}\vec{\boldsymbol{S}})\boldsymbol{y} = \boldsymbol{v}U\boldsymbol{p},$$

implying $\mu_{F_q}(\boldsymbol{v}\vec{\boldsymbol{S}}) \leq \boldsymbol{wt}(\boldsymbol{v}U)$. Since $\mu_{F_q}(\boldsymbol{v}\vec{\boldsymbol{S}}) = \operatorname{rank}(\boldsymbol{v}\vec{\boldsymbol{S}})$ and, by Proposition 6, $\operatorname{rank}(\boldsymbol{v}\vec{\boldsymbol{S}}) = \deg \boldsymbol{f}_{\boldsymbol{v}\vec{\boldsymbol{S}}}(\alpha)$, we have

$$\begin{aligned} t(q^k - q^{k-1}) &= \sum_{\boldsymbol{v} \in F^k} \boldsymbol{wt}(\boldsymbol{v}U) \\ &\geq \sum_{\boldsymbol{v} \in F^k} \mu_{F_q}(\boldsymbol{v}\vec{\boldsymbol{S}}) = \sum_{\boldsymbol{v} \in F^k} \deg \boldsymbol{f}_{\boldsymbol{v}\vec{\boldsymbol{S}}}(\alpha) = \sum_{H \in [\boldsymbol{S}]} \deg \boldsymbol{f}_H(\alpha), \end{aligned}$$

which implies the desired inequality. ◻

We conclude this section with an example that will be used for the proof of Theorem 1.

*Example* 16. Let

$$\epsilon_q = \frac{q^4}{q^5 + (q-1)^3} \tag{4}$$

and let $\boldsymbol{D}$ be the set of Hankel matrices defining the components of $W\boldsymbol{z}$ in (2). *If there exists a subset $\boldsymbol{D}'$ of $\boldsymbol{D}$ of cardinality $\lceil \lg_q \lg_q n - \lg_q \lg_q \lg_q n \rceil$ such that for each $H \in [\boldsymbol{D}']$, $\deg \boldsymbol{f}_H(\alpha) \geq (1 - \epsilon_q)n$, then*

$$\mu_{F_q}(n) \geq \left(3 + \frac{(q-1)^2}{q^5 + (q-1)^3}\right) n - o(n).$$

That is, Theorem 1 is true in this degenerate case.

Indeed, by Corollary 15,

$$\begin{aligned}\mu_{F_q}(\boldsymbol{D}') &\ge \frac{\sum\limits_{H\in[\boldsymbol{D}']}\deg \boldsymbol{f}_H(\alpha)}{q^{\lceil \lg_q \lg_q n - \lg_q \lg_q \lg_q n\rceil} - q^{\lceil \lg_q \lg_q n - \lg_q \lg_q \lg_q n\rceil - 1}}\\ &\ge \frac{(q^{\lceil \lg_q \lg_q n - \lg_q \lg_q \lg_q n\rceil} - 1)(1-\epsilon_q)n}{q^{\lceil \lg_q \lg_q n - \lg_q \lg_q \lg_q n\rceil} - q^{\lceil \lg_q \lg_q n - \lg_q \lg_q \lg_q n\rceil - 1}}\\ &\ge \left(1+\frac{1}{q-1}\right)(1-\epsilon_q)n - \frac{(1-\epsilon_q)n \lg_q \lg_q n}{\lg_q n}\end{aligned}$$

and the desired inequality follows from (4) and Proposition 2.

## 3. Limitations of Lemma 8 and its extension.

First we observe that Example 12 does not use much of the finiteness of the underlying field, because Lemma 8 holds for any field. In addition, Lemma 11 uses only "a half" of the dimension of $[\boldsymbol{D}]$ that equals $2n$. It seems that a better bound could be achieved by extending Lemma 8 "beyond $n+1$" (just in the case of a finite field), which would allow us to use a "bigger portion of $\dim([\boldsymbol{D}])$." As shown in Theorem 21 in section 3.2, Lemma 8 can indeed be extended, and we apply its extension for the proof of Theorem 1.

This section is organized as follows. First we present an example that indicates some limitations of Lemma 8. Then we state and prove Theorem 21 that extends Lemma 8.

### 3.1. Limitations of Lemma 8.

Our example illustrating some of the limitations of Lemma 8 is based on the following lemma.

LEMMA 17. *Let $H'$ and $H''$ be $(n+1)\times n$ Hankel matrices over $F_q$ and let $\alpha \in F_{q^2}\setminus F_q$.*[4] *Then each quadratic algorithm over $F_q$ that computes $\boldsymbol{x}^T(H'+\alpha H'')\boldsymbol{y}$ also computes $\boldsymbol{x}^T H'\boldsymbol{y}$ and $\boldsymbol{x}^T H''\boldsymbol{y}$.*

*Proof.* Let $\{(L_i'(\boldsymbol{x},\boldsymbol{y}), L_i''(\boldsymbol{x},\boldsymbol{y}))\}_{i=1,2,\ldots,t}$ be a quadratic algorithm over $F_q$ that computes $\boldsymbol{x}^T(H'+\alpha H'')\boldsymbol{y}$. That is, there exist $a_i, b_i \in F_q$, $i=1,2,\ldots,t$, such that

$$\begin{aligned}\boldsymbol{x}^T(H'+\alpha H'')\boldsymbol{y} &= \boldsymbol{x}^T H'\boldsymbol{y} + \alpha \boldsymbol{x}^T H''\boldsymbol{y}\\ &= \sum_{i=1}^t (a_i+\alpha b_i) L_i'(\boldsymbol{x},\boldsymbol{y}) L_i''(\boldsymbol{x},\boldsymbol{y})\\ &= \sum_{i=1}^t a_i L_i'(\boldsymbol{x},\boldsymbol{y}) L_i''(\boldsymbol{x},\boldsymbol{y}) + \alpha \sum_{i=1}^t b_i L_i'(\boldsymbol{x},\boldsymbol{y}) L_i''(\boldsymbol{x},\boldsymbol{y}).\end{aligned}$$

Since the coefficients of all $L_i'(\boldsymbol{x},\boldsymbol{y})$ and $L_i''(\boldsymbol{x},\boldsymbol{y})$, $i=1,2,\ldots,t$, belong to $F_q$, $\boldsymbol{x}^T H'\boldsymbol{y} = \sum_{i=1}^t a_i L_i'(\boldsymbol{x},\boldsymbol{y}) L_i''(\boldsymbol{x},\boldsymbol{y})$ and $\boldsymbol{x}^T H''\boldsymbol{y} = \sum_{i=1}^t b_i L_i'(\boldsymbol{x},\boldsymbol{y}) L_i''(\boldsymbol{x},\boldsymbol{y})$. ☐

*Remark* 18. A bit more involved argument shows that Lemma 17 holds for any two fields $F \subset F'$ and any $\alpha \in F'\setminus F$.

Now we are ready to present our example on which we base the proof of Theorem 1.

*Example* 19. Let $n$ be even and let $M, M'$, and $M''$ be $(n+1)\times n$ Hankel matrices over $F_q$ such that $\boldsymbol{f}_M(\alpha)$, $\boldsymbol{f}_{M'}(\alpha)$, and $\boldsymbol{f}_{M''}(\alpha)$ are pairwise coprime divisors of degree $\frac{n+1}{2}$. Below we prove that $\mu_{F_q}(\{M,M',M''\}) \ge (1+\frac{1}{2q^2})(n+1)$, whereas Lemma 8 gives us only $\mu_{F_q}(\{M,M',M''\}) \ge n+1$.

[4]Of course, we may assume that $F_q \subset F_{q^2}$.

Let $\{(L'_i(\boldsymbol{x},\boldsymbol{y}), L''_i(\boldsymbol{x},\boldsymbol{y}))\}_{i=1,2,\ldots,t}$ be a quadratic algorithm over $F_q$ that computes the bilinear forms defined by those three matrices. Then this algorithm also computes the bilinear forms over $F_{q^2}$ defined by $M+\alpha M'$ and $M''$, where $\alpha \in F_{q^2} \setminus F_q$ is as in Lemma 17. Let $\boldsymbol{p}$ be a column vector of products $L'_i(\boldsymbol{x},\boldsymbol{y})L''_i(\boldsymbol{x},\boldsymbol{y})$, $i=1,2,\ldots,t$. Then there exists a $2\times t$ matrix $U$ over $F_{q^2}$ such that

$$\left(\begin{array}{c} \boldsymbol{x}^T(M+\alpha M')\boldsymbol{y} \\ \boldsymbol{x}^T M''\boldsymbol{y} \end{array}\right) = U\boldsymbol{p}.$$

Therefore,

$$\boldsymbol{x}^T M''\boldsymbol{y} = (0,1)U\boldsymbol{p},$$

implying

$$\boldsymbol{wt}((0,1)U) \geq \mu_{F_q}(\{M''\}). \tag{5}$$

In addition, for $u,v \in F_q$,

$$\begin{aligned}\boldsymbol{x}^T((M+uM'')+\alpha(M'+vM''))\boldsymbol{y} &= \boldsymbol{x}^T((M+\alpha M')+(u+v\alpha)M'')\boldsymbol{y} \\ &= (1,u+v\alpha)U\boldsymbol{p},\end{aligned}$$

implying, by Lemma 17,

$$\boldsymbol{wt}((1,u+v\alpha)U) \geq \mu_{F_q}(\{M+uM'', M'+vM''\}). \tag{6}$$

Now, combining (5) and (6) with Corollary 14, we obtain

$$t \geq \frac{\mu_{F_q}(\{M''\}) + \displaystyle\sum_{u,v\in F_q} \mu_{F_q}(\{M+uM'', M'+vM''\})}{q^2}. \tag{7}$$

Since

$$\mu_{F_q}(\{M''\}) = \mathrm{rank}(M'') = \deg \boldsymbol{f}_{M''}(\alpha) = \frac{n+1}{2}$$

and, by Lemma 8, $\mu_{F_q}(\{M+rM'', M'+sM''\}) \geq n+1$, the desired inequality $t \geq (1+\frac{1}{2q^2})(n+1)$ follows from (7).

*Remark* 20. Replacing $\{M+\alpha M', M''\}$ in the above example with $\{M+\alpha M', M+\alpha M''\}$, one can show that $\mu_{F_q}(\{M,M',M''\}) \geq (1+\frac{1}{q^2})(n+1)$. However, we do not know how to generalize this approach.

**3.2. A generalization of Example 19.** Theorem 21 below (whose proof is presented in the next section) generalizes Example 19 by extending Lemma 8 "beyond $n+1$." We precede the statement of the theorem with the following notation.

For a set $\boldsymbol{H}$ of $(n+1)\times n$ Hankel matrices we denote by $\|\boldsymbol{H}\|$ the minimum of $\{\deg \boldsymbol{f}_H(\alpha) : H \in [\boldsymbol{H}] \setminus \{\boldsymbol{0}_{(n+1)\times n}\}$ and we denote by $\deg \boldsymbol{H}$ the minimum of $\{\deg \boldsymbol{f}_{\boldsymbol{H}'}(\alpha) : [\boldsymbol{H}'] = [\boldsymbol{H}]\}$.[5]

THEOREM 21. *Let $\boldsymbol{M}$ be a set of three linearly independent $(n+1)\times n$ Hankel matrices over $F_q$ such that $\deg \boldsymbol{f}_{\boldsymbol{M}}(\alpha) \geq n+1$.*

[5] Note the difference between $\deg \boldsymbol{H}$ and $\deg \boldsymbol{f}_{\boldsymbol{H}}(\alpha)$. Whereas the latter refers to a particular set of Hankel matrices $\boldsymbol{H}$, the former is the minimum over all sets $\boldsymbol{H}'$ which span $[\boldsymbol{H}]$. Nevertheless, by Lemma 10, they are equal, if either of them is less than $n+1$.

1. *If there is a basis* $\{M, M', M''\}$ *of* $[\boldsymbol{M}]$ *such that*

$$\deg \boldsymbol{f}_{\{M,M''\}}(\alpha), \deg \boldsymbol{f}_{\{M,M'\}}(\alpha), \deg \boldsymbol{f}_{\{M',M''\}}(\alpha) \le n, \tag{8}$$

*then*

$$\mu_{F_q}(\boldsymbol{M}) \ge n + 1 + \frac{\deg \boldsymbol{f}_{\{M,M',M''\}}(\alpha) - n - 1}{q^2}.$$

2. *If for each basis* $\{M, M', M''\}$ *of* $[\boldsymbol{M}]$ (8) *does not hold, then*

$$\mu_{F_q}(\boldsymbol{M}) \ge n + 1 + \frac{\|\boldsymbol{M}\|}{q^2}.$$

COROLLARY 22. *Let* $\boldsymbol{M}$ *be a set of three linearly independent* $(n+1) \times n$ *Hankel matrices over* $F_q$*. If* $\deg \boldsymbol{f}_{\boldsymbol{M}}(\alpha) \ge n+1$*, then*

$$\mu_{F_q}(\boldsymbol{M}) \ge n + 1 + \frac{\min(\deg \boldsymbol{M} - n - 1, \|\boldsymbol{M}\|)}{q^2}.$$

In view of Corollary 22, we shall search for a subset $\boldsymbol{D}'$ of $\boldsymbol{D}$, where $\boldsymbol{D}$ is the set of Hankel matrices defining the components of $W\boldsymbol{z}$ in (2), such that

- $|\boldsymbol{D}'| = o(n)$ and
- $[\boldsymbol{D}']$ includes a set $\boldsymbol{M}$ of three linearly independent $(n+1)\times n$ Hankel matrices satisfying

$$\min(\deg \boldsymbol{M} - n - 1, \|\boldsymbol{M}\|) \ge \frac{(q-1)^2}{q^2}\epsilon_q n,$$

where $\epsilon_q$ is defined by (4).

**3.3. Proof of Theorem 21.** We start with the proof of the first claim of the theorem. In this case we may assume that $\boldsymbol{M} = \{M, M', M''\}$, where $M$, $M'$, and $M''$ satisfy (8). Since $\mu_{F_q}(\{M''\}) = \deg \boldsymbol{f}_{M''}(\alpha)$, by (7), $\mu_{F_q}(\boldsymbol{M})$ is bound from below by

$$\frac{\deg \boldsymbol{f}_{M''}(\alpha) + \sum\limits_{u,v \in F} \mu_{F_q}(\{M + uM'', M' + vM''\})}{q^2}, \tag{9}$$

which we are going to estimate.

Let $\boldsymbol{f}_{\boldsymbol{M}}(\alpha) = \prod_{i=1}^{\ell} p_i^{d_i}(\alpha)$ be the decomposition of $\boldsymbol{f}_{\boldsymbol{M}}(\alpha)$ into irreducible factors. Then $M = \sum_{i=1}^{\ell} M_i$, $M' = \sum_{i=1}^{\ell} M_i'$, and $M'' = \sum_{i=1}^{\ell} M_i''$, where each of $\boldsymbol{f}_{M_i}(\alpha)$, $\boldsymbol{f}_{M_i'}(\alpha)$, and $\boldsymbol{f}_{M_i''}(\alpha)$ divides $p_i^{d_i}(\alpha)$, $i = 1, 2, \ldots, \ell$.[6] Thus,

$$M + uM'' = \sum_{i=1}^{\ell} (M_i + uM_i'')$$

and

$$M' + vM'' = \sum_{i=1}^{\ell} (M_i' + vM_i'').$$

We shall need the following property of matrices $M_i$, $M_i'$, and $M_i''$, $i = 1, 2, \ldots, \ell$.

PROPOSITION 23. *For each* $i = 1, 2, \ldots, \ell$

[6]Of course, some of $M_i$, $M_i'$, and $M_i''$, $i = 1, 2, \ldots, \ell$, may be $\mathbf{0}_{(n+1)\times n}$.

1. *inequality*

$$\boldsymbol{f}_{\{M_i+uM_i'',M_i'+vM_i''\}}(\alpha) \neq p_i^{d_i}(\alpha) \tag{10}$$

*implies* $\boldsymbol{f}_{M_i''}(\alpha) = p_i^{d_i}(\alpha)$, *and*

2. (10) *holds for at most one pair* $(u,v) \in F_q^2$.

*Proof.* Assume that $\boldsymbol{f}_{M_i''}(\alpha) \neq p_i^{d_i}(\alpha)$. Then, by the definition of $M_i$, $M_i'$, and $M_i''$, $\boldsymbol{f}_{\{M_i,M_i'\}}(\alpha) = p_i^{d_i}(\alpha)$, because $\boldsymbol{f}_{M_i''}(\alpha)$ properly divides $p_i^{d_i}(\alpha)$. However, $\boldsymbol{f}_{\{M_i,M_i'\}}(\alpha) = p_i^{d_i}(\alpha)$ together with $\boldsymbol{f}_{M_i''}(\alpha) \neq p_i^{d_i}(\alpha)$ (and Lemma 10) contradicts (10).

For the proof of the second claim, assume to the contrary that for two distinct pairs $(u_1,v_1)$ and $(u_2,v_2)$ neither of polynomials $\boldsymbol{f}_{\{M_i+u_1M_i'',M_i'+v_1M_i''\}}(\alpha)$ and $\boldsymbol{f}_{\{M_i+u_2M_i'',M_i'+v_2M_i''\}}(\alpha)$ equals $p_i^{d_i}(\alpha)$. We assume that $u_1 \neq u_2$ (the case of $v_1 \neq v_2$ is treated in a similar manner).

By the definition of $M_i$, $M_i'$, and $M_i''$, both $\boldsymbol{f}_{M_i+u_1M_i''}(\alpha)$ and $\boldsymbol{f}_{M_i+u_2M_i''}(\alpha)$ divide $p_i^{d_i-1}(\alpha)$. Therefore, by Lemma 10, $\boldsymbol{f}_{M''}(\alpha)$ also divides $p_i^{d_i-1}(\alpha)$, which contradicts the first claim of the proposition. □

Now, for a pair $(u,v) \in F_q^2$, let the set $\boldsymbol{I}_{(u,v)}$ consist of all integers $i = 1,2,\ldots,\ell$ which satisfy (10). Then

$$\begin{aligned}\deg \boldsymbol{f}_{\{M+uM'',M'+vM''\}}(\alpha) &= \sum_{i=1}^{\ell} \deg \boldsymbol{f}_{\{M_i+uM_i'',M_i'+vM_i''\}}(\alpha) \\ &\geq \deg \boldsymbol{f}_{\boldsymbol{M}}(\alpha) - \deg \prod_{i\in \boldsymbol{I}_{(u,v)}} p_i^{d_i}(\alpha),\end{aligned}$$

implying, by Lemmas 8 and 9,

$$\mu_{F_q}(\{M_i+uM_i'',M_i'+vM_i''\}) \geq \min\left(n+1, \deg \boldsymbol{f}_{\boldsymbol{M}}(\alpha) - \deg \prod_{i\in \boldsymbol{I}_{(u,v)}} p_i^{d_i}(\alpha)\right). \tag{11}$$

Therefore, summing over all pairs $(u,v) \in F_q^2$ and adding $\deg \boldsymbol{f}_{M''}(\alpha)$, we obtain

$$\begin{aligned}&\deg \boldsymbol{f}_{M''}(\alpha) + \sum_{(u,v)\in F_q^2} \mu_{F_q}(\{M+uM'',M'+vM''\}) \\ &\geq \deg \boldsymbol{f}_{M''}(\alpha) + \sum_{(u,v)\in F_q^2} \min\left(n+1, \deg \boldsymbol{f}_{\boldsymbol{M}}(\alpha) - \deg \prod_{i\in \boldsymbol{I}_{(u,v)}} p_i^{d_i}(\alpha)\right) \\ &\geq \deg \boldsymbol{f}_{M''}(\alpha) + (q^2-1)(n+1) \\ &\qquad + \min\left(n+1, \deg \boldsymbol{f}_{\boldsymbol{M}}(\alpha) - \sum_{(u,v)\in F_q^2} \deg \prod_{i\in \boldsymbol{I}_{(u,v)}} p_i^{d_i}(\alpha)\right) \\ &\geq \deg \boldsymbol{f}_{M''}(\alpha) + (q^2-1)(n+1) + \min(n+1, \deg \boldsymbol{f}_{\boldsymbol{M}}(\alpha) - \deg \boldsymbol{f}_{M''}(\alpha)) \\ &= \deg \boldsymbol{f}_{M''}(\alpha) + (q^2-1)(n+1) + (\deg \boldsymbol{f}_{\boldsymbol{M}}(\alpha) - \deg \boldsymbol{f}_{M''}(\alpha)) \\ &= q^2(n+1) + (\deg \boldsymbol{f}_{\boldsymbol{M}}(\alpha) - n - 1),\end{aligned}$$

where the first inequality follows from (11) and the second inequality follows from a trivial observation that for positive real numbers $a$, $b$, and $c$ such that $c \geq n+1$,

$$\min(n+1, c-a) + \min(n+1, c-b) \geq n+1+\min(n+1, c-(a+b)).$$

To prove the last inequality we observe that, by the second claim of Proposition 23, sets $\boldsymbol{I}_{(u,v)}$ are mutually disjoint. Thus, by the first claim of that proposition, $\prod_{(u,v)\in F_q^2}\prod_{i\in \boldsymbol{I}_{(u,v)}} p_i^{d_i}(\alpha)$ divides $\boldsymbol{f}_{M''}(\alpha)$.

Finally, the first equality follows from $\deg \boldsymbol{f}_{\boldsymbol{M}}(\alpha) - \deg \boldsymbol{f}_{M''}(\alpha) \leq n$. This is because $\deg \boldsymbol{f}_{\{M,M'\}}(\alpha) \leq n$ and $\boldsymbol{M} = \{M, M', M''\}$.

Now, the first claim of the theorem follows from (7).

For the proof of the second claim it suffices to find a basis $\{M, M', M''\}$ of $\boldsymbol{M}$ such that for all $u, v \in F_q$,

$$\mu_{F_q}(\{M+uM'', M'+vM''\}) \geq n+1. \tag{12}$$

This, together with (9), would imply

$$\mu_{F_q}(\boldsymbol{M}) \geq n+1+\frac{\deg \boldsymbol{f}_{M''}(\alpha)}{q^2},$$

which, in turn, would imply the second claim of the theorem.

So, let $\{M, M', M''\}$ be a basis of $[\boldsymbol{M}]$. If for all $u, v \in F_q$ it satisfies (12), we are done. Otherwise, it follows from Lemma 8 that for some elements $u$ and $v$ of $F_q$, $\deg \boldsymbol{f}_{\{M+uM'', M'+vM''\}}(\alpha) \leq n$, and we replace $M$, $M'$, and $M''$ with $M+uM''$, $M''$, and $M'+vM''$, respectively, which we shall also denote by $M$, $M'$, and $M''$, respectively. That is, we may assume that

$$\deg \boldsymbol{f}_{\{M,M''\}}(\alpha) \leq n. \tag{13}$$

Again, if for all $u, v \in F_q$, (12) holds (in the new basis), we are done.

Otherwise, for some $u, v \in F_q$, $\deg \boldsymbol{f}_{\{M+uM'', M'+vM''\}}(\alpha) \leq n$, and we replace $M$, $M'$, and $M''$ with $M'+vM''$, $M''$, and $M+uM''$, respectively, which we shall also denote by $M$, $M'$, and $M''$, respectively. Then, in view of (13), we may assume that

$$\deg \boldsymbol{f}_{\{M,M''\}}(\alpha), \deg \boldsymbol{f}_{\{M',M''\}}(\alpha) \leq n. \tag{14}$$

Now we contend that for all $u, v \in F_q$, (12) holds (in the new basis).

To prove our contention, assume to the contrary that for some $u, v \in F_q$, (12) does not hold. Then, by Lemma 8, $\deg \boldsymbol{f}_{\{M+uM'', M'+vM''\}}(\alpha) \leq n$, and we replace $M$ and $M'$ with $M+uM''$ and $M'+vM''$, respectively, which we shall also denote by $M$ and $M'$, respectively. Then, in view of (14),

$$\deg \boldsymbol{f}_{\{M,M''\}}(\alpha), \deg \boldsymbol{f}_{\{M,M'\}}(\alpha), \deg \boldsymbol{f}_{\{M',M''\}}(\alpha) \leq n,$$

in contradiction with the assumption of the second claim of the theorem.

**4. Outline of the proof of Theorem 1.** In this very short section we outline the proof of Theorem 1.

Let $\epsilon_q$ be as in (4) and let $\boldsymbol{D} = \{D_1, D_2, \ldots, D_{2n}\}$ be the set of Hankel matrices defining the components of $W\boldsymbol{z}$ in (2). In view of Example 16, we may assume that the linear closure of each subset of $\boldsymbol{D}$ of cardinality $\lceil \lg_q \lg_q n - \lg_q \lg_q \lg_q n \rceil$ contains an element $H$ such that $\deg \boldsymbol{f}_H(\alpha) < (1-\epsilon_q)n$. Under this assumption we shall construct a subset of $\boldsymbol{D}$ of cardinality $o(n)$ whose linear closure contains three Hankel matrices $M, M'$, and $M''$ such that

$$\min(\deg\{M, M', M''\} - n - 1, \|\{M, M', M''\}\|) \geq \frac{(q-1)^2}{q^2}\epsilon_q n. \tag{15}$$

This together with Corollary 22, Proposition 2, and (4) will prove Theorem 1.

The rest of the paper is organized as follows. In the next section we improve the known lower bound given by Lemma 9 and in section 6 we choose a small cardinality subset $\boldsymbol{D}'$ of $\boldsymbol{D}$ whose linear closure includes a set $\boldsymbol{S}$ of Hankel matrices possessing some special properties. The set $\{M, M', M''\}$ that satisfies (15) is chosen from $[\boldsymbol{S}]$. Matrices $M$ and $M'$ are constructed in section 7, and matrix $M''$ is constructed in section 8.

## 5. A coding-like lower bound, an extension of Lemma 9, and multiplication of polynomials modulo a polynomial.

In section 5.1 we prove a major technical lemma according to which linear spaces of Hankel matrices contain a matrix with an associated divisor of a (relatively) high degree and in section 5.2 we extend Lemma 9 and apply the extension to multiplication of polynomials modulo a polynomial.

### 5.1. A coding-like lower bound.

In this section we estimate the sum of the degrees of the divisors associated with all linear combinations of a set of Hankel matrices. We start with notation that will be used throughout the rest of this paper.

Let $\boldsymbol{S} = \{H_1, H_2, \ldots, H_k\}$ be a set of $(n+1)\times n$ Hankel matrices and let $\boldsymbol{f}_{\boldsymbol{S}}(\alpha) = \prod_{i=1}^{\ell} p_i^{d_i}(\alpha)$ be the decomposition of $\boldsymbol{f}_{\boldsymbol{S}}(\alpha)$ into irreducible factors. Then for each $H \in \boldsymbol{S}$ and each $i = 1, 2, \ldots, \ell$ there is the unique Hankel matrix $H|_i$—the $p_i(\alpha)$-component of $H$—such that $\boldsymbol{f}_{H|_i}(\alpha)$ divides $p_i^{d_i}(\alpha)$ and $H = \sum_{i=1}^{\ell} H|_i$; cf. the proof of Theorem 21. The set of all $p_i(\alpha)$-components of the elements of $\boldsymbol{S}$, $i = 1, 2, \ldots, \ell$, will be denoted by $\boldsymbol{S}|_i$. That is,

$$\boldsymbol{S}|_i = \{H|_i : H \in \boldsymbol{S}\}.$$

We shall denote by $\vec{\boldsymbol{S}}$ and $\vec{\boldsymbol{S}}|_i$, $i = 1, 2, \ldots, \ell$, the column vectors of the elements of $\boldsymbol{S}$ and $\boldsymbol{S}|_i$, respectively. That is, $\vec{\boldsymbol{S}} = (H_1, H_2, \ldots, H_k)^T$ and $\vec{\boldsymbol{S}}|_i = (H_1|_i, H_2|_i, \ldots, H_k|_i)^T$, say. Vector $\vec{\boldsymbol{S}}|_i$, $i = 1, 2, \ldots, \ell$, will be referred to as the $p_i(\alpha)$-*component* of $\vec{\boldsymbol{S}}$, and the set of all $p_i(\alpha)$-components of $\vec{\boldsymbol{S}}$, $i = 1, 2, \ldots, \ell$, will be denoted $\mathcal{C}(\vec{\boldsymbol{S}})$.

Next, for a vector $\boldsymbol{v} \in F_q^k$, the set of Hankel matrices $\{\boldsymbol{v}\vec{\boldsymbol{S}}|_i : i = 1, 2, \ldots, \ell\}$ will be denoted by $\boldsymbol{v}\mathcal{C}(\vec{\boldsymbol{S}})$. In this notation, $\boldsymbol{f}_{\boldsymbol{v}\mathcal{C}(\vec{\boldsymbol{S}})}(\alpha)$ is the product $\prod_{i=1}^{\ell} \boldsymbol{f}_{\boldsymbol{v}\vec{\boldsymbol{S}}|_i}(\alpha)$ of the divisors $\boldsymbol{f}_{\boldsymbol{v}\vec{\boldsymbol{S}}|_i}(\alpha)$ associated with the summands $\boldsymbol{v}\vec{\boldsymbol{S}}|_i$ of $\boldsymbol{v}\vec{\boldsymbol{S}}$, $i = 1, 2, \ldots, \ell$.

In particular, if each $\boldsymbol{f}_{H_i}(\alpha)$ is the minimal polynomial of an infinite linear recurring sequence $\sigma_i$ and $\boldsymbol{v} = (v_1, v_2, \ldots, v_k)$, then $\boldsymbol{f}_{\boldsymbol{v}\mathcal{C}(\vec{\boldsymbol{S}})}(\alpha)$ is the minimal polynomial of $\sum_{i=1}^{k} v_i\sigma_i$.

*Remark* 24. Note the difference between $\boldsymbol{f}_{\boldsymbol{v}\vec{\boldsymbol{S}}}(\alpha)$ and $\boldsymbol{f}_{\boldsymbol{v}\mathcal{C}(\vec{\boldsymbol{S}})}(\alpha)$. The degree of the former is at most $n$, whereas the degree of the latter can exceed it. The divisors are equal if and only if $\deg \boldsymbol{f}_{\boldsymbol{v}\mathcal{C}(\vec{\boldsymbol{S}})}(\alpha) \leq n$.

Proposition 25. *For each $\boldsymbol{v}', \boldsymbol{v}'' \in F_q^k$,*

$$\begin{aligned}\deg \boldsymbol{f}_{\boldsymbol{v}'\mathcal{C}(\vec{\boldsymbol{S}})}(\alpha) + \deg \boldsymbol{f}_{\boldsymbol{v}''\mathcal{C}(\vec{\boldsymbol{S}})}(\alpha) &\geq \deg \operatorname{lcm}\{\boldsymbol{f}_{\boldsymbol{v}'\mathcal{C}(\vec{\boldsymbol{S}})}(\alpha), \boldsymbol{f}_{\boldsymbol{v}''\mathcal{C}(\vec{\boldsymbol{S}})}(\alpha)\}\\ &\geq \deg \boldsymbol{f}_{(\boldsymbol{v}'+\boldsymbol{v}'')\mathcal{C}(\vec{\boldsymbol{S}})}(\alpha).\end{aligned}$$

*Proof.* Since, by definition, $\boldsymbol{f}_{\boldsymbol{v}\mathcal{C}(\vec{\boldsymbol{S}})}(\alpha) = \prod_{i=1}^{\ell} \boldsymbol{f}_{\boldsymbol{v}\vec{\boldsymbol{S}}|_i}(\alpha)$, $i = 1, 2, \ldots, \ell$, it suffices to show that

$$\begin{aligned}\deg \boldsymbol{f}_{\boldsymbol{v}'\vec{\boldsymbol{S}}|_i}(\alpha) + \deg \boldsymbol{f}_{\boldsymbol{v}''\vec{\boldsymbol{S}}|_i}(\alpha) &\geq \deg \operatorname{lcm}\{\boldsymbol{f}_{\boldsymbol{v}'\vec{\boldsymbol{S}}|_i}(\alpha), \boldsymbol{f}_{\boldsymbol{v}''\vec{\boldsymbol{S}}|_i}(\alpha)\}\\ &\geq \deg \boldsymbol{f}_{(\boldsymbol{v}'+\boldsymbol{v}'')\vec{\boldsymbol{S}}|_i}(\alpha).\end{aligned}$$

The first inequality is trivial and the second inequality holds because, by the definition of $\boldsymbol{S}|_i$, one of the divisors $\boldsymbol{f}_{\boldsymbol{v}'\vec{\boldsymbol{S}}|_i}(\alpha)$ and $\boldsymbol{f}_{\boldsymbol{v}''\vec{\boldsymbol{S}}|_i}(\alpha)$ divides the other. □

Finally, for divisors $\boldsymbol{f}(\alpha)$ and $\boldsymbol{g}(\alpha)$ we denote by $\deg_{\boldsymbol{g}(\alpha)} \boldsymbol{f}(\alpha)$ the degree of $\frac{\operatorname{lcm}\{\boldsymbol{g}(\alpha), \boldsymbol{f}(\alpha)\}}{\boldsymbol{g}(\alpha)}$. That is,

$$\deg_{\boldsymbol{g}(\alpha)} \boldsymbol{f}(\alpha) = \deg \operatorname{lcm}\{\boldsymbol{g}(\alpha), \boldsymbol{f}(\alpha)\} - \deg \boldsymbol{g}(\alpha). \tag{16}$$

Proposition 26 (cf. Proposition 13). *Let $\boldsymbol{S}$ be a k-element set of $(n+1) \times n$ Hankel matrices. Then*

$$\sum_{\boldsymbol{v} \in F_q^k} \deg_{\boldsymbol{g}(\alpha)} \boldsymbol{f}_{\boldsymbol{v}\mathcal{C}(\vec{\boldsymbol{S}})}(\alpha) \geq (q^k - q^{k-1}) \deg_{\boldsymbol{g}(\alpha)} \boldsymbol{f}_{\boldsymbol{S}}(\alpha). \tag{17}$$

*Proof.* Let $\boldsymbol{f}_{\boldsymbol{S}}(\alpha) = \prod_{i=1}^{\ell} p_i^{d_i}(\alpha)$. It follows from the definition of $\boldsymbol{f}_{\boldsymbol{v}\mathcal{C}(\vec{\boldsymbol{S}})}(\alpha)$ and (16) that

$$\deg_{\boldsymbol{g}(\alpha)} \boldsymbol{f}_{\boldsymbol{v}\mathcal{C}(\vec{\boldsymbol{S}})}(\alpha) = \sum_{i=1}^{\ell} \deg_{\boldsymbol{g}(\alpha)} \boldsymbol{f}_{\boldsymbol{v}\vec{\boldsymbol{S}}|_i}(\alpha).$$

Therefore,

$$\sum_{\boldsymbol{v} \in F_q^k} \deg_{\boldsymbol{g}(\alpha)} \boldsymbol{f}_{\boldsymbol{v}\mathcal{C}(\vec{\boldsymbol{S}})}(\alpha) = \sum_{i=1}^{\ell} \sum_{\boldsymbol{v} \in F_q^k} \deg_{\boldsymbol{g}(\alpha)} \boldsymbol{f}_{\boldsymbol{v}\vec{\boldsymbol{S}}|_i}(\alpha), \tag{18}$$

and we start with the calculation of $\sum_{\boldsymbol{v} \in F_q^k} \deg_{\boldsymbol{g}(\alpha)} \boldsymbol{f}_{\boldsymbol{v}\vec{\boldsymbol{S}}|_i}(\alpha)$.

Let

$$\boldsymbol{S}_{i,j} = \{H \in [\boldsymbol{S}|_i] : \boldsymbol{f}_H(\alpha) | p_i^j(\alpha)\}, \tag{19}$$

where $i = 1, 2, \ldots, \ell$ and $j = 0, 1, \ldots, d_i$. Then $\boldsymbol{v}\vec{\boldsymbol{S}}|_i \in \boldsymbol{S}_{i,j} \setminus \boldsymbol{S}_{i,j-1}$ implies

$$\deg_{\boldsymbol{g}(\alpha)} \boldsymbol{f}_{\boldsymbol{v}\vec{\boldsymbol{S}}|_i}(\alpha) = \deg_{\boldsymbol{g}(\alpha)} p_i^j(\alpha). \tag{20}$$

First, for each $j = 0, 1, \ldots, d_i$ we shall calculate the number of vectors $\boldsymbol{v} \in F_q^k$ such that $\boldsymbol{v}\vec{\boldsymbol{S}}|_i \in \boldsymbol{S}_{i,j}$ or, equivalently,

$$\boldsymbol{v}\vec{\boldsymbol{S}}|_i \equiv \boldsymbol{0}_{(n+1)\times n} \bmod \boldsymbol{S}_{i,j}. \tag{21}$$

Since the dimension of the vector space of solutions of (21) is

$$k - \dim(\boldsymbol{S}_{i,d_i}/\boldsymbol{S}_{i,j}) = k - (\dim \boldsymbol{S}_{i,d_i} - \dim \boldsymbol{S}_{i,j}),$$

where, as usual, $\boldsymbol{S}_{i,d_i}/\boldsymbol{S}_{i,j}$ denotes the quotient vector space of $\boldsymbol{S}_{i,d_i}$ modulo $\boldsymbol{S}_{i,j}$, the number of vectors $\boldsymbol{v}$ satisfying (21) is $q^{k-(\dim \boldsymbol{S}_{i,d_i} - \dim \boldsymbol{S}_{i,j})}$. Therefore, the number of vectors $\boldsymbol{v}$ such that $\boldsymbol{v}\vec{\boldsymbol{S}}|_i \in \boldsymbol{S}_{i,j} \setminus \boldsymbol{S}_{i,j-1}$ is

$$q^{k-(\dim \boldsymbol{S}_{i,d_i} - \dim \boldsymbol{S}_{i,j})} - q^{k-(\dim \boldsymbol{S}_{i,d_i} - \dim \boldsymbol{S}_{i,j-1})},$$

and it follows from (18) and (20) that

$$\begin{aligned} &\sum_{\boldsymbol{v}\in F_q^k} \deg_{\boldsymbol{g}(\alpha)} \boldsymbol{f}_{\boldsymbol{v}\mathcal{C}(\vec{\boldsymbol{S}})}(\alpha) \\ &= \sum_{i=1}^{\ell}\sum_{j=1}^{d_i} \deg_{\boldsymbol{g}(\alpha)} p_i^j(\alpha)\left(q^{k-(\dim \boldsymbol{S}_{i,d_i} - \dim \boldsymbol{S}_{i,j})} - q^{k-(\dim \boldsymbol{S}_{i,d_i} - \dim \boldsymbol{S}_{i,j-1})}\right). \end{aligned} \tag{22}$$

With no loss of generality, replacing $\boldsymbol{g}(\alpha)$ with $\gcd(\boldsymbol{f}_{\boldsymbol{S}}(\alpha), \boldsymbol{g}(\alpha))$ if necessary, we may assume that $\boldsymbol{g}(\alpha)$ divides $\boldsymbol{f}_{\boldsymbol{S}}(\alpha)$. That is, $\boldsymbol{g}(\alpha) = \prod_{i=1}^{\ell} p_i^{e_i}(\alpha)$, where $e_i \le d_i$, $i = 1, 2, \ldots, \ell$. Then for $i = 1, 2, \ldots, \ell$ and a nonnegative integer $j$,

$$\deg_{\boldsymbol{g}(\alpha)} p_i^j(\alpha) = \begin{cases} \deg p_i^{j-e_i}(\alpha) & \text{if } j \ge e_i, \\ 0 & \text{otherwise.} \end{cases}$$

Therefore, by (22), the left-hand side of (17) equals

$$\begin{aligned} &\sum_{i=1}^{\ell} \deg p_i(\alpha) \sum_{j=e_i+1}^{d_i} (j-e_i)\left(q^{k-(\dim \boldsymbol{S}_{i,d_i} - \dim \boldsymbol{S}_{i,j})} - q^{k-(\dim \boldsymbol{S}_{i,d_i} - \dim \boldsymbol{S}_{i,j-1})}\right) \\ &= \sum_{i=1}^{\ell} q^{k-\dim \boldsymbol{S}_{i,d_i}} \deg p_i(\alpha) \sum_{j=e_i+1}^{d_i} (j-e_i)\left(q^{\dim \boldsymbol{S}_{i,j}} - q^{\dim \boldsymbol{S}_{i,j-1}}\right). \end{aligned} \tag{23}$$

Since

$$\begin{aligned} &\sum_{j=e_i+1}^{d_i} (j-e_i)\left(q^{\dim \boldsymbol{S}_{i,j}} - q^{\dim \boldsymbol{S}_{i,j-1}}\right) \\ &= \sum_{j=e_i+1}^{d_i} (j-e_i) q^{\dim \boldsymbol{S}_{i,j}} - \sum_{j=e_i+1}^{d_i} (j-e_i) q^{\dim \boldsymbol{S}_{i,j-1}} \\ &= \sum_{j=e_i+1}^{d_i} (j-e_i) q^{\dim \boldsymbol{S}_{i,j}} - \sum_{j=e_i}^{d_i-1} (j-e_i+1) q^{\dim \boldsymbol{S}_{i,j}} \\ &= (d_i - e_i) q^{\dim \boldsymbol{S}_{i,d_i}} - \sum_{j=e_i}^{d_i-1} q^{\dim \boldsymbol{S}_{i,j}}, \end{aligned}$$

(23) equals

$$\begin{aligned} &\sum_{i=1}^{\ell} q^{k-\dim \boldsymbol{S}_{i,d_i}} \deg p_i(\alpha) \left( (d_i-e_i) q^{\dim \boldsymbol{S}_{i,d_i}} - \sum_{j=e_i}^{d_i-1} q^{\dim \boldsymbol{S}_{i,j}} \right) \\ &= q^k \deg_{\boldsymbol{g}(\alpha)} \boldsymbol{f}_{\boldsymbol{S}}(\alpha) - q^{k-1} \sum_{i=1}^{\ell} \deg p_i(\alpha) \sum_{j=e_i}^{d_i-1} \frac{q}{q^{\dim \boldsymbol{S}_{i,d_i} - \dim \boldsymbol{S}_{i,j}}}. \end{aligned} \tag{24}$$

Now, adding to (24) the zero sum

$$-q^{k-1}\deg_{\boldsymbol{g}(\alpha)}\boldsymbol{f}_{\boldsymbol{S}}(\alpha)+q^{k-1}\sum_{i=1}^{\ell}(d_i-e_i)\deg p_i(\alpha),$$

we obtain

$$\begin{aligned}&\sum_{\boldsymbol{v}\in F_q^k}\deg_{\boldsymbol{g}(\alpha)}\boldsymbol{f}_{\boldsymbol{v}\mathcal{C}(\vec{\boldsymbol{S}})}(\alpha)\\ &=(q^k-q^{k-1})\deg_{\boldsymbol{g}(\alpha)}\boldsymbol{f}_{\boldsymbol{S}}(\alpha)\\ &\quad+q^{k-1}\sum_{i=1}^{\ell}\left((d_i-e_i)-\sum_{j=e_i}^{d_i-1}\frac{q}{q^{\dim\boldsymbol{S}_{i,d_i}-\dim\boldsymbol{S}_{i,j}}}\right)\deg p_i(\alpha).\end{aligned}\tag{25}$$

Since for $j<d_i$, $\dim\boldsymbol{S}_{i,d_i}>\dim\boldsymbol{S}_{i,j}$, we have

$$(d_i-e_i)-\sum_{j=e_i}^{d_i-1}\frac{q}{q^{\dim\boldsymbol{S}_{i,d_i}-\dim\boldsymbol{S}_{i,j}}}\geq 0,$$

which together with (25) completes the proof. ☐

We conclude this section with an example that will be used in the proof of Theorem 1.

*Example* 27. Let $\boldsymbol{S}=\{H_1,H_2,\ldots,H_k\}$. *If for all* $\boldsymbol{v}\in F_q^k$,

$$\deg\boldsymbol{f}_{\boldsymbol{v}\mathcal{C}(\vec{\boldsymbol{S}})}(\alpha)\leq n,\tag{26}$$

*then* $\mu_{F_q}(\boldsymbol{S})\geq\deg\boldsymbol{f}_{\boldsymbol{S}}(\alpha)$.

Indeed,

$$\mu_{F_q}(\boldsymbol{S})\geq\frac{\sum_{\boldsymbol{v}\in F_q^k}\deg\boldsymbol{f}_{\boldsymbol{v}\vec{\boldsymbol{S}}}(\alpha)}{q^k-q^{k-1}}=\frac{\sum_{\boldsymbol{v}\in F_q^k}\deg\boldsymbol{f}_{\boldsymbol{v}\mathcal{C}(\vec{\boldsymbol{S}})}(\alpha)}{q^k-q^{k-1}}\geq\deg\boldsymbol{f}_{\boldsymbol{S}}(\alpha),$$

where the first inequality is by Corollary 15 (because $[\boldsymbol{S}]=\{\boldsymbol{v}\vec{\boldsymbol{S}}:\boldsymbol{v}\in F_q^k\}$), the equality is by (26) and Remark 24, and the last inequality follows from Proposition 26 with $\boldsymbol{g}(\alpha)=1$.

## 5.2. An extension of Lemma 9 and multiplication of polynomials modulo a polynomial.

We start with the following extension of Lemma 9.

LEMMA 28. *Let* $\boldsymbol{S}$ *be a set of* $(n+1)\times n$ *Hankel matrices. Let* $\boldsymbol{f}_{\boldsymbol{S}}(\alpha)=\prod_{i=1}^{\ell}p_i^{d_i}(\alpha)$ *be the decomposition of* $\boldsymbol{f}_{\boldsymbol{S}}(\alpha)$ *into irreducible factors and let* $\boldsymbol{S}_{i,j}$, $i=1,2,\ldots,\ell$, $j=0,1,\ldots,d_i$, *be as in the proof of Proposition* 26. *If* $\deg\boldsymbol{f}_{\boldsymbol{S}}(\alpha)\leq n$, *then*

$$\mu_{F_q}(\boldsymbol{S})\geq\left(1+\frac{1}{q-1}\right)\deg\boldsymbol{f}_{\boldsymbol{S}}(\alpha)-\frac{1}{q-1}\sum_{i=1}^{\ell}\left(\sum_{j=0}^{d_i-1}\frac{q}{q^{\dim\boldsymbol{S}_{i,d_i}-\dim\boldsymbol{S}_{i,j}}}\right)\deg p_i(\alpha).$$

*Proof.* Since $\deg\boldsymbol{f}_{\boldsymbol{S}}(\alpha)\leq n$, by Lemma 10 and Remark 24, $\boldsymbol{f}_{\boldsymbol{v}\mathcal{C}(\vec{\boldsymbol{S}})}(\alpha)=\boldsymbol{f}_{\boldsymbol{v}\vec{\boldsymbol{S}}}(\alpha)$ for each $\boldsymbol{v}\in F_q^k$. Therefore, the lemma follows from Corollary 15, (25) with $\boldsymbol{g}(\alpha)=1$,[7] and equality $\deg\boldsymbol{f}_{\boldsymbol{S}}(\alpha)=\sum_{i=1}^{\ell}d_i\deg p_i(\alpha)$. ☐

[7]Of course, in both Corollary 15 and (25), $k=|\boldsymbol{S}|$.

In particular, as an easy corollary to Lemma 28, we obtain a more intuitive and direct proof of the lower bound on the complexity of multiplication of polynomials modulo a polynomial ([4] and [7]) with a better lower order term.

We shall need the following notation. Let $P(\alpha)$ be a polynomial of degree $n$ over $F_q$, and let $\boldsymbol{C}_{P(\alpha)}$ be the set of the coefficients of the minimal degree residue of $\sum_{k=0}^{2n-1} z_k \alpha^k$ modulo $P(\alpha)$, where $z_k$s are defined by (1).

Corollary 29 (cf. [4] and [7, Corollary 4.5]). *We have*

$$\mu_{F_q}(\boldsymbol{C}_{P(\alpha)}) \geq \left(2 + \frac{1}{q-1}\right) n - o(n).$$

*Proof.* Assume that $\mu_{F_q}(\boldsymbol{C}_{P(\alpha)}) = t$. That is, there exist $t$ pairs of linear forms $(L_1'(\boldsymbol{x},\boldsymbol{y}), L_1''(\boldsymbol{x},\boldsymbol{y})), \ldots, (L_t'(\boldsymbol{x},\boldsymbol{y}), L_t''(\boldsymbol{x},\boldsymbol{y}))$ such that the elements of $\boldsymbol{C}_{P(\alpha)}$ are linear combinations of products $L_i'(\boldsymbol{x},\boldsymbol{y})L_i''(\boldsymbol{x},\boldsymbol{y})$, $i = 1,2,\ldots,t$. Let $\boldsymbol{p}$ be a column vector of the above products. Since $\dim[\boldsymbol{C}_{P(\alpha)}] = n$, there exists an $n \times t$ matrix $U$ such that $\vec{\boldsymbol{C}}_{P(\alpha)} = U\boldsymbol{p}$, where $\vec{\boldsymbol{C}}_{P(\alpha)}$ is a column vector of the elements of $\boldsymbol{C}_{P(\alpha)}$.

Permuting the components of $\boldsymbol{p}$ if necessary, we may assume that the first $n$ columns of $U$ are linearly independent. Hence there exist a nonsingular $n \times n$ matrix $W$ and an $n \times (t-n)$ matrix $V$ such that

$$W\vec{\boldsymbol{C}}_{P(\alpha)} = (\boldsymbol{I}_n, V)\boldsymbol{p}.$$

Let $\boldsymbol{C}^W$ denote the set of the components of $W\vec{\boldsymbol{C}}_{P(\alpha)}$. By [15, Lemma 6], $\boldsymbol{f}_{\boldsymbol{C}_{P(\alpha)}}(\alpha) = P(\alpha)$. Therefore, by Lemma 10, $\boldsymbol{f}_{\boldsymbol{C}^W}(\alpha) = P(\alpha)$ as well, implying that $\boldsymbol{C}^W$ is a basis of $\{H : \boldsymbol{f}_H(\alpha) | P(\alpha)\}$, because the elements of $\boldsymbol{C}^W$ are linearly independent.

Let $P(\alpha) = \prod_{i=1}^{\ell} p_i^{d_i}(\alpha)$ be the decomposition of $P(\alpha)$ into irreducible factors and let $k_i = \lceil \lg_q \deg p_i^{d_i}(\alpha) \rceil$, $i = 1,2,\ldots,\ell$.

For each $i = 1,2,\ldots,\ell$, let $\boldsymbol{C}_i$ be a $k_i$-element subset of $\boldsymbol{C}^W$ such that the elements of $\boldsymbol{C}_i|_i$ are linearly independent modulo $[\boldsymbol{C}_{p_i^{d_i - \lfloor k_i / \deg p_i(\alpha) \rfloor - 1}}(\alpha)]$ and $\boldsymbol{C}_i|_i$ contains a basis of $[\boldsymbol{C}^W|_i]$ modulo $[\boldsymbol{C}_{p_i^{d_i - \lfloor k_i / \deg p_i(\alpha) \rfloor}}(\alpha)]$.

Let $\boldsymbol{S} = \bigcup_{i=1}^{\ell} \boldsymbol{C}_i$. Similarly to [1, Proposition 1], one can show that

$$\mu_{F_q}(\boldsymbol{C}^W) \geq n + \mu_{F_q}(\boldsymbol{S}) - |\boldsymbol{S}|. \tag{27}$$

Since $[\boldsymbol{C}_{P(\alpha)}] = [\boldsymbol{C}^W]$, $\mu_{F_q}(\boldsymbol{C}_{P(\alpha)}) = \mu_{F_q}(\boldsymbol{C}^W)$, and it follows from the definition of $\boldsymbol{S}$ that $|\boldsymbol{S}| \leq \sum_{i=1}^{\ell} k_i$. This together with (27) implies

$$\mu_{F_q}(\boldsymbol{C}_{P(\alpha)}) \geq n + \mu_{F_q}(\boldsymbol{S}) - \sum_{i=1}^{\ell} k_i.$$

By the definition of $\boldsymbol{C}_i$, $\boldsymbol{f}_{\boldsymbol{C}_i}(\alpha) = p_i^{d_i}(\alpha)$, $i = 1,2,\ldots,\ell$. Therefore,

$$\deg \boldsymbol{f}_{\boldsymbol{S}}(\alpha) = \sum_{i=1}^{\ell} d_i \deg p_i(\alpha) = \deg P(\alpha) = n,$$

which together with Lemma 28 implies

$$\mu_{F_q}(\boldsymbol{S}) \geq \left(1 + \frac{1}{q-1}\right) n - \frac{q}{q-1} \sum_{i=1}^{\ell} \left( \sum_{j=0}^{d_i - 1} \frac{1}{q^{\dim \boldsymbol{S}_{i,d_i} - \dim \boldsymbol{S}_{i,j}}} \right) \deg p_i(\alpha).$$

Hence it suffices to show that

$$\sum_{i=1}^{\ell}\left(\frac{q}{q-1}\left(\sum_{j=0}^{d_i-1}\frac{1}{q^{\dim \boldsymbol{S}_{i,d_i}-\dim \boldsymbol{S}_{i,j}}}\right)\deg p_i(\alpha)+k_i\right) \tag{28}$$

is $o(n)$.

It follows from the definition of $\boldsymbol{S}$ that

- $\dim \boldsymbol{S}_{i,d_i}-\dim \boldsymbol{S}_{i,j}=(d_i-j)\deg p_i(\alpha)$ if $j\ge d_i-\lfloor k_i/\deg p_i(\alpha)\rfloor$ and
- $\dim \boldsymbol{S}_{i,d_i}-\dim \boldsymbol{S}_{i,j}\ge k_i$ otherwise.

Therefore,

$$\begin{aligned}\sum_{j=0}^{d_i-1}\frac{1}{q^{\dim \boldsymbol{S}_{i,d_i}-\dim \boldsymbol{S}_{i,j}}} &\le \sum_{j=1}^{\lfloor k_i/\deg p_i(\alpha)\rfloor}\frac{1}{q^{j\deg p_i(\alpha)}}+\frac{d_i-\lfloor k_i/\deg p_i(\alpha)\rfloor}{q^{k_i}}\\ &= \frac{1}{q^{\deg p_i(\alpha)}}\frac{1-q^{-\lfloor k_i/\deg p_i(\alpha)\rfloor\deg p_i(\alpha)}}{1-q^{-\deg p_i(\alpha)}}+\frac{d_i-\lfloor k_i/\deg p_i(\alpha)\rfloor}{q^{k_i}}\\ &\le \frac{1}{q^{\deg p_i(\alpha)}-1}+\frac{d_i}{q^{k_i}}.\end{aligned}$$

Consequently,

$$\left(\sum_{j=0}^{d_i-1}\frac{1}{q^{\dim \boldsymbol{S}_{i,d_i}-\dim \boldsymbol{S}_{i,j}}}\right)\deg p_i(\alpha)\le \frac{\deg p_i(\alpha)}{q^{\deg p_i(\alpha)}-1}+\frac{d_i\deg p_i(\alpha)}{q^{k_i}}\le\frac{q+1}{q-1},$$

because

$$\frac{\deg p_i(\alpha)}{q^{\deg p_i(\alpha)}-1}\le\frac{1}{q}$$

and, by the definition of $k_i$,

$$q^{k_i}\ge d_i\deg p_i(\alpha).$$

Therefore,

$$\frac{q}{q-1}\left(\sum_{j=0}^{d_i-1}\frac{1}{q^{\dim \boldsymbol{S}_{i,d_i}-\dim \boldsymbol{S}_{i,j}}}\right)\deg p_i(\alpha)+k_i\le\frac{q(q+1)}{(q-1)^2}+k_i,$$

which, in turn, implies that (28) is bound by

$$\begin{aligned}\frac{q(q+1)}{(q-1)^2}\ell+\sum_{i=1}^{\ell}k_i &\le \frac{q(q+1)}{(q-1)^2}\ell+\sum_{i=1}^{\ell}\lg_q\deg p_i^{d_i}(\alpha)+\ell\\ &\le \frac{q(q+1)}{(q-1)^2}i_q(n)+\lg_q\deg\prod_{i=1}^{\ell}p_i^{d_i}(\alpha)+i_q(n)\\ &= \frac{2q^2-q+1}{(q-1)^2}i_q(n)+\lg_q n,\end{aligned}$$

which completes the proof. □

*Remark* 30. A straightforward inspection shows that [15, Lemma 6] extends to divisors. That is, for a divisor $P(\alpha)$, $\boldsymbol{f}_{\boldsymbol{C}_{P(\alpha)}}(\alpha)=P(\alpha)$.[8] Therefore, Corollary 29 holds for computation of the polynomial product modulo a divisor as well.

[8]See [1, section 2] for the definition of the minimal degree residue modulo a divisor.

## 6. Proof of Theorem 1. Part 1: Construction of a subset of $\boldsymbol{D}$ of a small cardinality.

First, in section 6.1 we prove a number of auxiliary results needed for our constructions. Then, in section 6.2 we choose a small cardinality subset $\boldsymbol{D}'$ of $\boldsymbol{D}$ whose linear closure includes a set of Hankel matrices $\boldsymbol{S}$ needed for the construction of $M$, $M'$, and $M''$.

### 6.1. An extension of Lemma 10.

In this section we describe divisors of some elements of the vector space spanned by a set of Hankel matrices.

PROPOSITION 31 (cf. Lemma 10). *Let $\boldsymbol{S}$ be a set of $(n+1)\times n$ Hankel matrices and let $H \in [\boldsymbol{S}]$. If $\deg \boldsymbol{f}_{\boldsymbol{S}\cup\{H\}}(\alpha) \le 2n$, then $\boldsymbol{f}_H(\alpha)$ divides $\boldsymbol{f}_{\boldsymbol{S}}(\alpha)$.*

The proof of Proposition 31 is based on its particular cases treated by Lemmas 32 and 33 below.

LEMMA 32. *Let $H'$ and $H''$ be $(n+1)\times n$ Hankel matrices such that $\boldsymbol{f}_{H'}(\alpha)$ and $\boldsymbol{f}_{H''}(\alpha)$ are coprime and*

$$\deg \boldsymbol{f}_{H'}(\alpha) + \deg \boldsymbol{f}_{H''}(\alpha) + \deg \boldsymbol{f}_{H'+H''}(\alpha) \le 2n. \tag{29}$$

*Then $\boldsymbol{f}_{H'+H''}(\alpha)$ divides $\boldsymbol{f}_{H'}(\alpha)\boldsymbol{f}_{H''}(\alpha)$.*

*Proof.* We shall denote $H' + H''$ by $H$.

Let $H$, $H'$, and $H''$ be defined by sequences $\sigma = s_0, s_1, \ldots, s_{2n-1}$, $\sigma' = s'_0, s'_1, \ldots, s'_{2n-1}$, and $\sigma'' = s''_0, s''_1, \ldots, s''_{2n-1}$, respectively.

Since $\boldsymbol{f}_{H'}(\alpha)$ and $\boldsymbol{f}_{H''}(\alpha)$ are coprime, we may assume that $\bar{H}''$ is the zero $(n+1)\times n$ matrix $\mathbf{0}_{(n+1)\times n}$.[9]

We contend that $\bar{H} = \bar{H}'$. By [18, Theorem 8.55, p. 425], $\boldsymbol{f}_{\tilde{H}'}(\alpha)\boldsymbol{f}_{\tilde{H}''}(\alpha)\boldsymbol{f}_{\tilde{H}}(\alpha)$ is a characteristic polynomial of $\tilde{\sigma}' + \tilde{\sigma}'' - \tilde{\sigma}$, and it follows from (29) that

$$\deg \boldsymbol{f}_{\tilde{H}'}(\alpha)\boldsymbol{f}_{\tilde{H}''}(\alpha)\boldsymbol{f}_{\tilde{H}}(\alpha) \le 2n - \max\{\mathrm{rank}(\bar{H}), \mathrm{rank}(\bar{H}')\}. \tag{30}$$

In addition, since

$$\tilde{H}' + \tilde{H}'' - \tilde{H} = (H' - \bar{H}') + (H'' - \bar{H}'') - (H - \bar{H}) = \bar{H} - \bar{H}', \tag{31}$$

the first $2n - \max\{\mathrm{rank}(\bar{H}), \mathrm{rank}(\bar{H}')\}$ elements of $\tilde{\sigma}' + \tilde{\sigma}'' - \tilde{\sigma}$ are 0. Thus, by (30), all elements of $\tilde{\sigma}' + \tilde{\sigma}'' - \tilde{\sigma}$ are 0 as well. This together with (31) implies

$$\tilde{H}' + \tilde{H}'' - \tilde{H} = \bar{H} - \bar{H}' = \mathbf{0}_{(n+1)\times n},$$

which, in turn, implies the desired equality $\bar{H} = \bar{H}'$.

It remains to show that $\boldsymbol{f}_{\tilde{H}}(\alpha)$ divides $\boldsymbol{f}_{\tilde{H}'}(\alpha)\boldsymbol{f}_{\tilde{H}''}(\alpha)$. Let

$$\mathrm{rank}(\bar{H}') = d \quad (= \mathrm{rank}(\bar{H}))$$

and let $\sigma|_{2n-d-1}$, $\sigma'|_{2n-d-1}$, and $\sigma''|_{2n-d-1}$ consist of the first $2n-d$ elements of $\sigma$, $\sigma'$, and $\sigma''$, respectively. Then $\boldsymbol{f}_{\tilde{H}'}(\alpha)$ and $\boldsymbol{f}_{\tilde{H}''}(\alpha)$ are characteristic polynomials of $\sigma'|_{2n-d-1}$ and $\sigma''|_{2n-d-1}$, respectively, and, by [18, Theorem 8.55, p. 425], $\boldsymbol{f}_{\tilde{H}'}(\alpha)\boldsymbol{f}_{\tilde{H}''}(\alpha)$ is a characteristic polynomial of $\sigma'|_{2n-d-1} + \sigma''|_{2n-d-1}$.

Since $\sigma'|_{2n-d-1} + \sigma''|_{2n-d-1} = \sigma|_{2n-d-1}$ and $\boldsymbol{f}_{\tilde{H}}(\alpha)$ is the minimal polynomial of $\sigma|_{2n-d-1}$, the result follows from Proposition 3 (that applies in view of (29)). ∎

LEMMA 33. *Let $\boldsymbol{S}$ be a set of $(n+1)\times n$ Hankel matrices and let $H \in [\boldsymbol{S}]$. If $\deg \boldsymbol{f}_H(\alpha) + \deg \boldsymbol{f}_{\boldsymbol{S}}(\alpha) \le 2n$, then $\boldsymbol{f}_H(\alpha)$ divides $\boldsymbol{f}_{\boldsymbol{S}}(\alpha)$.*

[9]See Definition 4 and Example 5.

*Proof.* Let $\boldsymbol{f}_{\boldsymbol{S}}(\alpha) = \prod_{i=1}^{\ell} p_i^{d_i}(\alpha)$ be the decomposition of $\boldsymbol{f}_{\boldsymbol{S}}(\alpha)$ into irreducible factors. Then $H = \sum_{i=1}^{\ell} H_i$, where $\boldsymbol{f}_{H_i}(\alpha)$ divides $p_i^{d_i}(\alpha)$, $i = 1, 2, \ldots, \ell$.

Since $\prod_{i=1}^{\ell} \boldsymbol{f}_{H_i}(\alpha)$ divides $\prod_{i=1}^{\ell} p_i^{d_i}(\alpha) = \boldsymbol{f}_{\boldsymbol{S}}(\alpha)$, it suffices to show that $\boldsymbol{f}_H(\alpha)$ divides $\prod_{i=1}^{\ell} \boldsymbol{f}_{H_i}(\alpha)$.

A straightforward induction based on Lemma 32 shows that $\boldsymbol{f}_{\sum_{i=1}^{k} H_i}(\alpha)$ divides $\prod_{i=1}^{k} \boldsymbol{f}_{H_i}(\alpha)$, $k = 2, 3, \ldots, \ell$, and the result follows with $k = \ell$. ◻

*Proof of Proposition* 31. Let $H = H' + H''$, where $\boldsymbol{f}_{H'}(\alpha)$ divides $\boldsymbol{f}_{\boldsymbol{S}}(\alpha)$ and $\boldsymbol{f}_{H''}(\alpha)$ is coprime with $\boldsymbol{f}_{\boldsymbol{S}}(\alpha)$.

Since $H \in [\boldsymbol{S}]$, $H'' \in [\boldsymbol{S} \cup \{H'\}]$. Therefore,

$$\deg \boldsymbol{f}_{\boldsymbol{S} \cup \{H'\}}(\alpha) + \deg \boldsymbol{f}_{H''}(\alpha) = \deg \boldsymbol{f}_{\boldsymbol{S} \cup \{H\}}(\alpha) \leq 2n.$$

Thus, by Lemma 33, $\boldsymbol{f}_{H''}(\alpha)$ divides $\boldsymbol{f}_{\boldsymbol{S} \cup \{H'\}}(\alpha)$, which together with the definition of $H''$ implies $H'' = \mathbf{0}_{(n+1)\times n}$. That is, $H = H'$, which completes the proof. ◻

**6.2. Construction of $\boldsymbol{S}$.** Let $\boldsymbol{D} = \{D_1, D_2, \ldots, D_{2n}\}$ be the set of Hankel matrices defining the components of $W\boldsymbol{z}$ in (2). Lemma 34 below provides us with a small cardinality subset $\boldsymbol{D}'$ of $\boldsymbol{D}$ and a subset $\boldsymbol{S}$ of $[\boldsymbol{D}']$ needed for the construction of three matrices $M$, $M'$, and $M''$ satisfying (15).

LEMMA 34. *Assume that the linear closure of each subset of $\boldsymbol{D}$ of cardinality $\lceil \lg_q \lg_q n - \lg_q \lg_q \lg_q n \rceil$ contains a matrix $H$ such that $\deg \boldsymbol{f}_H(\alpha) < (1 - \epsilon_q)n$. Then there exists a subset $\boldsymbol{D}'$ of $\boldsymbol{D}$ of cardinality at most $\frac{(1+\epsilon_q)n \lg_q \lg_q n}{\lg_q((1+\epsilon_q)n)}$ and there exists a subset $\boldsymbol{S}$ of $[\boldsymbol{D}']$ such that*

1. *for each $H \in \boldsymbol{S}$, $\deg \boldsymbol{f}_H(\alpha) < (1 - \epsilon_q)n$ and*
2. *$\deg \boldsymbol{f}_{\boldsymbol{S}}(\alpha) \geq (1 + \epsilon_q)n$.*

*Proof.* For a $\lceil \lg_q \lg_q n - \lg_q \lg_q \lg_q n \rceil$-element subset $\boldsymbol{C}$ of $\boldsymbol{D}$, let $H_{\boldsymbol{C}} \in [\boldsymbol{C}]$ be such that $\deg \boldsymbol{f}_{H_{\boldsymbol{C}}}(\alpha) < (1 - \epsilon_q)n$ and let

$$\boldsymbol{H} = \{H_{\boldsymbol{C}} : |\boldsymbol{C}| = \lceil \lg_q \lg_q n - \lg_q \lg_q \lg_q n \rceil\}. \tag{32}$$

We contend that $\dim[\boldsymbol{H}] \geq 2n - \lceil \lg_q \lg_q n - \lg_q \lg_q \lg_q n \rceil$. For the proof, assume to the contrary that $\dim[\boldsymbol{H}] < 2n - \lceil \lg_q \lg_q n - \lg_q \lg_q \lg_q n \rceil$. Let $\vec{\boldsymbol{D}}$ be a column vector of the elements of $\boldsymbol{D}$ and let $U$ be a $\dim[\boldsymbol{H}] \times 2n$ matrix such that the components of $U\vec{\boldsymbol{D}}$ form a basis of $[\boldsymbol{H}]$. Permuting the components of $\vec{\boldsymbol{D}}$ and applying linear transformations if necessary, we may assume that $U$ is of the form $(\boldsymbol{I}_{\dim[\boldsymbol{H}]}, V)$. Then

$$[\boldsymbol{H}] \cap [\{D_{2n-i} : i = 0, 1, \ldots, \lceil \lg_q \lg_q n - \lg_q \lg_q \lg_q n \rceil - 1\}] = \emptyset.$$

Therefore,

$$H_{\{D_{2n-i} : i=0,1,\ldots,\lceil \lg_q \lg_q n - \lg_q \lg_q \lg_q n \rceil - 1\}} \notin \boldsymbol{H},$$

in contradiction with the definition of $\boldsymbol{H}$. This proves our contention.

Since

$$2n - \lceil \lg_q \lg_q n - \lg_q \lg_q \lg_q n \rceil > (1 + \epsilon_q)n,$$

by Lemma 11, there exists a subset $\boldsymbol{S}$ of $\boldsymbol{H}$ of cardinality at most $i_q((1 + \epsilon_q)n)$ such that $\deg \boldsymbol{f}_{\boldsymbol{S}}(\alpha) \geq (1 + \epsilon_q)n$.

It remains to construct a small cardinality subset $\boldsymbol{D}'$ of $\boldsymbol{D}$ such that $\boldsymbol{S} \subseteq [\boldsymbol{D}']$.

For each element $H$ of $\boldsymbol{S}$, let $\boldsymbol{C}_H$ be a $\lceil \lg_q \lg_q n - \lg_q \lg_q \lg_q n \rceil$-element subset of $\boldsymbol{D}$ such that $H = H_{\boldsymbol{C}_H}$ and let

$$\boldsymbol{D}' = \bigcup_{H \in \boldsymbol{S}} \boldsymbol{C}_H.$$

Then $\boldsymbol{S} \subset [\boldsymbol{D}']$ and

$$|\boldsymbol{D}'| \leq \lceil \lg_q \lg_q n - \lg_q \lg_q \lg_q n \rceil i_q((1+\epsilon_q)n) \leq \frac{(1+\epsilon_q)n \lg_q \lg_q n}{\lg_q((1+\epsilon_q)n)},$$

which completes the proof. □

COROLLARY 35. *Let $\boldsymbol{S}$ be as in Lemma* 34. *Then* $\deg \boldsymbol{S} \geq (1+\epsilon_q)n$.

*Proof.* Assume to the contrary that for some basis $\boldsymbol{S}'$ of $[\boldsymbol{S}]$, $\deg \boldsymbol{f}_{\boldsymbol{S}'}(\alpha) < (1+\epsilon_q)n$. Let $H \in \boldsymbol{S}$. Then $H \in [\boldsymbol{S}']$, because $\boldsymbol{S}'$ is a basis of $[\boldsymbol{S}]$. Also, by Lemma 34, $\deg \boldsymbol{f}_H(\alpha) < (1-\epsilon_q)n$, implying

$$\deg \boldsymbol{f}_{\boldsymbol{S}'}(\alpha) + \deg \boldsymbol{f}_H(\alpha) < (1+\epsilon_q)n + (1-\epsilon_q)n = 2n.$$

Therefore, by Lemma 33, $\boldsymbol{f}_H(\alpha)$ divides $\boldsymbol{f}_{\boldsymbol{S}'}(\alpha)$, and, consequently, $\boldsymbol{f}_{\boldsymbol{S}}(\alpha)$ divides $\boldsymbol{f}_{\boldsymbol{S}'}(\alpha)$. However, this contradicts our assumption $\deg \boldsymbol{f}_{\boldsymbol{S}'}(\alpha) < (1+\epsilon_q)n$, because $\deg \boldsymbol{f}_{\boldsymbol{S}}(\alpha) \geq (1+\epsilon_q)n$. □

Let $\boldsymbol{S}$ be as in Lemma 34. Deleting some elements from $\boldsymbol{S}$ if necessary, we may assume that for no proper subset $\boldsymbol{S}'$ of $\boldsymbol{S}$, $\deg \boldsymbol{f}_{\boldsymbol{S}'}(\alpha) \geq (1+\epsilon_q)n$. Also, replacing $\boldsymbol{H}$ defined in the proof of Lemma 34 with its maximal linearly independent subset, we may assume that the elements of $\boldsymbol{S}$ are linearly independent.

## 7. Proof of Theorem 1. Part 2: Construction of $M$ and $M'$.

Let $\boldsymbol{S}$ be the set of matrices constructed in section 6.2. In this section we construct Hankel matrices $M$ and $M'$, $M, M' \in [\boldsymbol{S}]$, such that

$$\deg \boldsymbol{f}_M(\alpha) < \left(1 - \frac{(q-1)^2}{q^2}\epsilon_q\right)n, \tag{33}$$

$$\deg \boldsymbol{f}_{M'}(\alpha) < \left(1 - \frac{(q-1)^2}{q^2}\epsilon_q\right)n, \tag{34}$$

$$\deg \boldsymbol{f}_{\{M,M'\}}(\alpha) \geq \left(1 - \frac{(q-1)^2}{q^2}\epsilon_q\right)n, \tag{35}$$

and

$$\|\{M, M'\}\| \geq \frac{(q-1)^2}{q^2}\epsilon_q n. \tag{36}$$

The construction is as follows. Let $\boldsymbol{S} = \{H_1, H_2, \ldots, H_k\}$. In view of Example 27 and Proposition 2, we may assume that for some $\boldsymbol{v} \in F_q^k$,

$$\deg \boldsymbol{f}_{\boldsymbol{v}\mathcal{C}(\vec{\boldsymbol{S}})}(\alpha) > n, \tag{37}$$

and we fix such a vector $\boldsymbol{v} = (v_1, v_2, \ldots, v_k)$ through the end of this section.

The first two matrices $M$ and $M'$ are appropriate subsums of $\boldsymbol{v}\vec{\boldsymbol{S}}$, and the third matrix $M''$ is constructed from the quotient vector space $[\boldsymbol{S}]/[M, M']$.

To construct $M$ and $M'$, which satisfy (33)–(36), we shall distinguish between the cases of

$$\max\{\deg \boldsymbol{f}_{H_i}(\alpha) : v_i \neq 0\} < \frac{(q-1)^2}{q^2}\epsilon_q n \tag{38}$$

and

$$\max\{\deg \boldsymbol{f}_{H_i}(\alpha) : v_i \neq 0\} \geq \frac{(q-1)^2}{q^2}\epsilon_q n \tag{39}$$

and use the following notation.

For $I \subseteq \{1, 2, \ldots, k\}$ we define a $k$-dimensional vector $\boldsymbol{v}^I = (v_1^I, v_2^I, \ldots, v_k^I)$ by

$$v_i^I = \begin{cases} v_i & \text{if } i \in I, \\ 0 & \text{otherwise,} \end{cases} \qquad i = 1, 2, \ldots, k.$$

**7.1. The case of $\max\{\deg \boldsymbol{f}_{H_i}(\alpha) : v_i \neq 0\} < \frac{(q-1)^2}{q^2}\epsilon_q n$.** Let $J$ be a minimal (with respect to inclusion) subset of $\{1, 2, \ldots, k\}$ such that

$$\deg \boldsymbol{f}_{\boldsymbol{v}^J \mathcal{C}(\vec{\boldsymbol{S}})}(\alpha) > \left(1 - \frac{(q-1)^2}{q^2}\epsilon_q\right) n \tag{40}$$

and let $I$ be a minimal (with respect to inclusion) subset of $J$ such that

$$\deg \boldsymbol{f}_{\boldsymbol{v}^I \mathcal{C}(\vec{\boldsymbol{S}})}(\alpha) \geq \frac{(q-1)^2}{q^2}\epsilon_q n. \tag{41}$$

Proposition 36. *We have*

$$\deg \boldsymbol{f}_{\boldsymbol{v}^I \mathcal{C}(\vec{\boldsymbol{S}})}(\alpha) < 2\left(\frac{(q-1)^2}{q^2}\epsilon_q n\right).$$

*Proof.* Assume to the contrary that

$$\deg \boldsymbol{f}_{\boldsymbol{v}^I \mathcal{C}(\vec{\boldsymbol{S}})}(\alpha) \geq 2\left(\frac{(q-1)^2}{q^2}\epsilon_q n\right). \tag{42}$$

Then, for each $i \in I$,

$$\begin{aligned}\deg \boldsymbol{f}_{\boldsymbol{v}^{I\setminus\{i\}} \mathcal{C}(\vec{\boldsymbol{S}})}(\alpha) &\geq \deg \boldsymbol{f}_{\boldsymbol{v}^I \mathcal{C}(\vec{\boldsymbol{S}})}(\alpha) - \deg \boldsymbol{f}_{\boldsymbol{v}^{\{i\}} \mathcal{C}(\vec{\boldsymbol{S}})}(\alpha) \\ &\geq 2\left(\frac{(q-1)^2}{q^2}\epsilon_q n\right) - \frac{(q-1)^2}{q^2}\epsilon_q n = \frac{(q-1)^2}{q^2}\epsilon_q n,\end{aligned}$$

where the first inequality is by Proposition 25 and the last inequality follows from (42) and (38). However,

$$\deg \boldsymbol{f}_{\boldsymbol{v}^{I\setminus\{i\}} \mathcal{C}(\vec{\boldsymbol{S}})}(\alpha) \geq \frac{(q-1)^2}{q^2}\epsilon_q n$$

contradicts the definition of $I$ as a minimal subset of $J$ satisfying (41). $\square$

By Proposition 36 and Remark 24, $\deg \boldsymbol{f}_{\boldsymbol{v}^I \mathcal{C}(\vec{\boldsymbol{S}})}(\alpha) = \deg \boldsymbol{f}_{\boldsymbol{v}^I \vec{\boldsymbol{S}}}(\alpha)$, and we let $M$ be $\boldsymbol{v}^I \vec{\boldsymbol{S}}$. Then (33) follows from Proposition 36 and (4).

Let $I'$ be a minimal (with respect to inclusion) subset of $J \setminus I$ such that

$$\deg \operatorname{lcm}\{\boldsymbol{f}_{\boldsymbol{v}^I \mathcal{C}(\vec{\boldsymbol{S}})}(\alpha), \boldsymbol{f}_{\boldsymbol{v}^{I'} \mathcal{C}(\vec{\boldsymbol{S}})}(\alpha)\} \geq \left(1 - \frac{(q-1)^2}{q^2}\epsilon_q\right) n. \tag{43}$$

The existence of $I'$ follows from (40) and Proposition 25.

It follows from the minimality assumption on $J$ that

$$\deg \boldsymbol{f}_{\boldsymbol{v}^{I'} \mathcal{C}(\vec{\boldsymbol{S}})}(\alpha) < \left(1 - \frac{(q-1)^2}{q^2}\epsilon_q\right) n. \tag{44}$$

Therefore, by Remark 24,

$$\deg \boldsymbol{f}_{\boldsymbol{v}^{I'} \mathcal{C}(\vec{\boldsymbol{S}})}(\alpha) = \deg \boldsymbol{f}_{\boldsymbol{v}^{I'} \vec{\boldsymbol{S}}}(\alpha)$$

and we let $M'$ be $\boldsymbol{v}^{I'} \vec{\boldsymbol{S}}$. Then (34) follows from (44), and (35) follows from (43).

For the proof of (36) we need Proposition 37 below.

Proposition 37. *We have*

$$\deg \boldsymbol{f}_{M'}(\alpha) \geq \left(1 - 3\left(\frac{(q-1)^2}{q^2}\epsilon_q\right)\right) n.$$

*Proof.* Were

$$\deg \boldsymbol{f}_{M'}(\alpha) = \deg \boldsymbol{f}_{\boldsymbol{v}^{I'} \mathcal{C}(\vec{\boldsymbol{S}})}(\alpha) < \left(1 - 3\left(\frac{(q-1)^2}{q^2}\epsilon_q\right)\right) n, \tag{45}$$

by Propositions 36 and 25, we would have

$$\begin{aligned}\left(1 - \frac{(q-1)^2}{q^2}\epsilon_q\right) n &= 2\left(\frac{(q-1)^2}{q^2}\epsilon_q n\right) + \left(1 - 3\left(\frac{(q-1)^2}{q^2}\epsilon_q\right)\right) n \\ &> \deg \boldsymbol{f}_{\boldsymbol{v}^I \mathcal{C}(\vec{\boldsymbol{S}})}(\alpha) + \deg \boldsymbol{f}_{\boldsymbol{v}^{I'} \mathcal{C}(\vec{\boldsymbol{S}})}(\alpha) \\ &\geq \deg \operatorname{lcm}\{\boldsymbol{f}_{\boldsymbol{v}^I \mathcal{C}(\vec{\boldsymbol{S}})}(\alpha), \boldsymbol{f}_{\boldsymbol{v}^{I'} \mathcal{C}(\vec{\boldsymbol{S}})}(\alpha)\},\end{aligned}$$

which contradicts (43). $\square$

In view of (41), for the proof of (36), it suffices to show that for all $v \in F_q$, $\deg \boldsymbol{f}_{vM+M'}(\alpha) \geq \frac{(q-1)^2}{q^2}\epsilon_q n$.

This is indeed so, because

$$\begin{aligned}\deg \boldsymbol{f}_{vM+M'}(\alpha) &\geq \deg \boldsymbol{f}_{M'}(\alpha) - \deg \boldsymbol{f}_M(\alpha) \\ &\geq \left(1 - 3\left(\frac{(q-1)^2}{q^2}\epsilon_q\right)\right) n - 2\left(\frac{(q-1)^2}{q^2}\epsilon_q\right) n \\ &\geq \frac{(q-1)^2}{q^2}\epsilon_q n,\end{aligned}$$

where the first inequality follows from Proposition 25, the second inequality follows from Propositions 37 and 36, and the last inequality follows from (4).

This completes the construction of $M$ and $M'$ in the case of (38).

We conclude this section with the observation that

$$\deg \boldsymbol{f}_{\{M,M'\}}(\alpha) < n. \tag{46}$$

This observation will be used in the construction of matrix $M''$ in section 8.1.2.

For the proof of (46), assume to the contrary that

$$\deg \boldsymbol{f}_{\{M,M'\}}(\alpha) = \deg \operatorname{lcm}\{\boldsymbol{f}_{\boldsymbol{v}^I\mathcal{C}(\vec{\boldsymbol{S}})}(\alpha), \boldsymbol{f}_{\boldsymbol{v}^{I'}\mathcal{C}(\vec{\boldsymbol{S}})}(\alpha)\} \geq n.$$

This together with (38) and (4) implies that for each $i \in I'$,

$$\begin{aligned}&\deg \operatorname{lcm}\{\boldsymbol{f}_{\boldsymbol{v}^I\mathcal{C}(\vec{\boldsymbol{S}})}(\alpha), \boldsymbol{f}_{\boldsymbol{v}^{I'\setminus\{i\}}\mathcal{C}(\vec{\boldsymbol{S}})}(\alpha)\}\\ &\geq \deg \operatorname{lcm}\{\boldsymbol{f}_{\boldsymbol{v}^I\mathcal{C}(\vec{\boldsymbol{S}})}(\alpha), \boldsymbol{f}_{\boldsymbol{v}^{I'}\mathcal{C}(\vec{\boldsymbol{S}})}(\alpha)\} - \deg \boldsymbol{f}_{H_i}(\alpha)\\ &\geq n - \frac{(q-1)^2}{q^2}\epsilon_q n = \left(1 - \frac{(q-1)^2}{q^2}\epsilon_q\right) n,\end{aligned}$$

which contradicts the definition of $I'$ as a minimal subset of $J \setminus I$ satisfying (43).

**7.2. The case of $\max\{\deg \boldsymbol{f}_{H_i}(\alpha) : v_i \neq 0\} \geq \frac{(q-1)^2}{q^2}\epsilon_q n$.** Let $i = 1, 2, \ldots, k$ be such that $\deg \boldsymbol{f}_{H_i}(\alpha) \geq \frac{(q-1)^2}{q^2}\epsilon_q n$ and let $I$ be a maximal (with respect to inclusion) subset of $\{1, 2, \ldots, k\}$ containing $i$ such that

$$\deg \boldsymbol{f}_{\boldsymbol{v}^I\mathcal{C}(\vec{\boldsymbol{S}})}(\alpha) < \left(1 - \frac{(q-1)^2}{q^2}\epsilon_q\right) n. \tag{47}$$

By Remark 24, $\deg \boldsymbol{f}_{\boldsymbol{v}^I\mathcal{C}(\vec{\boldsymbol{S}})}(\alpha) = \deg \boldsymbol{f}_{\boldsymbol{v}^I\vec{\boldsymbol{S}}}(\alpha)$, and we let $M$ be $\boldsymbol{v}^I\vec{\boldsymbol{S}}$. Then (33) immediately follows from the definition.

To construct $M'$ we shall distinguish between the cases of

$$\max\{\deg \boldsymbol{f}_{H_i}(\alpha) : i \in \{1, 2, \ldots, k\} \setminus I\} \geq \frac{(q-1)^2}{q^2}\epsilon_q n \tag{48}$$

and

$$\max\{\deg \boldsymbol{f}_{H_i}(\alpha) : i \in \{1, 2, \ldots, k\} \setminus I\} < \frac{(q-1)^2}{q^2}\epsilon_q n. \tag{49}$$

If (48), let $M'$ be an element of $\{H_i : i \in \{1, 2, \ldots, k\} \setminus I\}$ such that

$$\deg \boldsymbol{f}_{M'}(\alpha) \geq \frac{(q-1)^2}{q^2}\epsilon_q n. \tag{50}$$

Then (34) immediately follows from the first claim of Lemma 34.

Otherwise, i.e., if (49), let $I'$ be a minimal (with respect to inclusion) subset of $\{1, 2, \ldots, k\} \setminus I$ such that

$$\deg \boldsymbol{f}_{\boldsymbol{v}^{I'}\mathcal{C}(\vec{\boldsymbol{S}})}(\alpha) \geq \frac{(q-1)^2}{q^2}\epsilon_q n. \tag{51}$$

The existence of $I'$ follows from (37), (47), and Proposition 25. Similarly to the proof of Proposition 36 one can show that

$$\deg \boldsymbol{f}_{\boldsymbol{v}^{I'}\mathcal{C}(\vec{\boldsymbol{S}})}(\alpha) < 2\left(\frac{(q-1)^2}{q^2}\epsilon_q n\right). \tag{52}$$

Therefore, by Remark 24, $\deg \boldsymbol{f}_{\boldsymbol{v}^{I'}\mathcal{C}(\vec{\boldsymbol{S}})}(\alpha) = \deg \boldsymbol{f}_{\boldsymbol{v}^{I'}\vec{\boldsymbol{S}}}(\alpha)$ and we let $M'$ be $\boldsymbol{v}^{I'}\vec{\boldsymbol{S}}$. Then (34) follows from (52) and (4), and (35) follows from the definition of $I$ in the beginning of this section.

It remains to prove (36). In view of (50) and (51), for the proof it suffices to show that for all $v \in F_q$, $\deg \boldsymbol{f}_{M+vM'}(\alpha) \geq \frac{(q-1)^2}{q^2}\epsilon_q n$.

So, assume to the contrary that for some $v \in F_q$, $\deg \boldsymbol{f}_{M+vM'}(\alpha) < \frac{(q-1)^2}{q^2}\epsilon_q n$. By Corollary 35, with no loss of generality, we may assume that $\boldsymbol{S}$ is a basis of $[\boldsymbol{S}]$ with the maximal number of elements $H$ such that

$$\deg \boldsymbol{f}_H(\alpha) < \frac{(q-1)^2}{q^2}\epsilon_q n. \tag{53}$$

However, $(\boldsymbol{S} \setminus \{H_i\}) \cup \{M + vM'\}$, where $i$ is as in the definition of $I$ in the beginning of this section, is a basis of $[\boldsymbol{S}]$ with more matrices $H$ satisfying (53), which contradicts our assumption.

**8. Proof of Theorem 1. Part 3: Construction of $M''$.** As we have already mentioned in the beginning of section 7, $M''$ is constructed from the quotient vector space $[\boldsymbol{S}]/[M, M']$. We remind the reader that $\boldsymbol{S} = \{H_1, H_2, \ldots, H_k\}$ is the set of Hankel matrices constructed in section 6.2. In view of Corollary 35, we may assume that

$$\{H_1, H_2\} = \{M, M'\}, \tag{54}$$

where $M$ and $M'$ are the matrices constructed in section 7.

Also, changing the indices of the elements of $\boldsymbol{S}$ if necessary, we may assume that

$$\begin{aligned}&\deg \boldsymbol{f}_{\boldsymbol{S}}(\alpha) - \deg \boldsymbol{f}_{\{H_1,H_2,\ldots,H_{k-1}\}}(\alpha)\\ &= \min\{\deg \boldsymbol{f}_{\boldsymbol{S}}(\alpha) - \deg \boldsymbol{f}_{\boldsymbol{S}\setminus\{H\}}(\alpha) : H \in \boldsymbol{S} \setminus \{H_1, H_2\}\}.\end{aligned}$$

Then, for each $i = 2, 3, \ldots, k-1$,

$$\begin{aligned}&\deg \boldsymbol{f}_{\boldsymbol{S}}(\alpha) - \deg \boldsymbol{f}_{\boldsymbol{S}\setminus\{H_k\}}(\alpha)\\ &\leq \deg \boldsymbol{f}_{\boldsymbol{S}}(\alpha) - \deg \boldsymbol{f}_{\boldsymbol{S}\setminus\{H_{i+1}\}}(\alpha)\\ &\leq \deg \boldsymbol{f}_{\{H_1,H_2,\ldots,H_i,H_{i+1}\}}(\alpha) - \deg \boldsymbol{f}_{\{H_1,H_2,\ldots,H_i\}}(\alpha),\end{aligned}$$

because each factor of $\boldsymbol{f}_{H_{i+1}}(\alpha)$ that does not divide $\boldsymbol{f}_{\boldsymbol{S}\setminus\{H_{i+1}\}}(\alpha)$ does not divide $\boldsymbol{f}_{\{H_1,H_2,\ldots,H_i\}}(\alpha)$ either. Therefore,

$$\begin{aligned}&(k-3)(\deg \boldsymbol{f}_{\boldsymbol{S}}(\alpha) - \deg \boldsymbol{f}_{\{H_1,H_2,\ldots,H_{k-1}\}}(\alpha))\\ &\leq \sum_{i=2}^{k-2}(\deg \boldsymbol{f}_{\{H_1,H_2,\ldots,H_i,H_{i+1}\}}(\alpha) - \deg \boldsymbol{f}_{\{H_1,H_2,\ldots,H_i\}}(\alpha))\\ &= \deg \boldsymbol{f}_{\boldsymbol{S}\setminus\{H_k\}}(\alpha) - \deg \boldsymbol{f}_{\{H_1,H_2\}}(\alpha),\end{aligned}$$

implying

$$\deg \boldsymbol{f}_{\boldsymbol{S}}(\alpha) \leq \deg \boldsymbol{f}_{\boldsymbol{S}\setminus\{H_k\}}(\alpha) + \frac{\deg \boldsymbol{f}_{\boldsymbol{S}\setminus\{H_k\}}(\alpha) - \deg \boldsymbol{f}_{\{H_1,H_2\}}(\alpha)}{k-3}. \tag{55}$$

For the construction of the third matrix $M''$ we shall distinguish between the cases of $k \geq 5$ and $k \leq 4$.

**8.1. The case of $\boldsymbol{k \geq 5}$.** By the minimality assumption on $\boldsymbol{S}$ in the end of section 6.2,

$$\deg \boldsymbol{f}_{\boldsymbol{S}\setminus\{H_k\}}(\alpha) \leq (1+\epsilon_q)n$$

and, by (54) and (35),

$$\deg \boldsymbol{f}_{\{H_1,H_2\}}(\alpha) \geq \left(1 - \frac{(q-1)^2}{q^2}\epsilon_q\right) n.$$

Therefore, it follows from (55) that

$$\deg \boldsymbol{f}_{\boldsymbol{S}}(\alpha) \leq (1+\epsilon_q)n + \frac{\epsilon_q + \dfrac{(q-1)^2}{q^2}\epsilon_q}{k-3} n,$$

which together with (4) and $k \geq 5$ implies

$$\deg \boldsymbol{f}_{\boldsymbol{S}}(\alpha) \leq \left(2 - 2\left(\frac{(q-1)^2}{q^2}\epsilon_q\right)\right) n. \tag{56}$$

*Remark* 38. Actually, (56) is the only reason we need the condition $k \geq 5$; see also section 8.2.3.

The construction of the third matrix $M''$ is based on Lemma 39 below.

LEMMA 39. *Let $\boldsymbol{S}$ be the set of matrices constructed in section* 6.2*, let $M$ and $M'$ be the matrices constructed in section* 7*, and let $\boldsymbol{v}'' \in F_q^k$ be such that*

$$\deg_{\boldsymbol{f}_{\{M,M'\}}(\alpha)} \boldsymbol{f}_{\boldsymbol{v}''\mathcal{C}(\vec{\boldsymbol{S}})}(\alpha) \geq \left(1 - \frac{1}{q}\right) \deg_{\boldsymbol{f}_{\{M,M'\}}(\alpha)} \boldsymbol{f}_{\boldsymbol{S}}(\alpha). \tag{57}$$

*Then*

$$\deg\{M, M', \boldsymbol{v}''\vec{\boldsymbol{S}}\} \geq \left(1 + \frac{(q-1)^2}{q^2}\epsilon_q\right) n. \tag{58}$$

*Moreover, if* $\deg \boldsymbol{f}_{\{M,M'\}}(\alpha) \geq n$*, then*

$$\deg\{M, M', \boldsymbol{v}''\vec{\boldsymbol{S}}\} \geq \left(1 + \frac{q-1}{q}\epsilon_q\right) n. \tag{59}$$

*Proof.* For the proof of the first part of the lemma, assume to the contrary that for some basis $\boldsymbol{B}$ of $[\{M, M', \boldsymbol{v}''\vec{\boldsymbol{S}}\}]$,

$$\deg \boldsymbol{f}_{\boldsymbol{B}}(\alpha) < \left(1 + \frac{(q-1)^2}{q^2}\epsilon_q\right) n. \tag{60}$$

Then it follows from (33), (34), and Lemma 33 that $\boldsymbol{f}_{\{M,M'\}}(\alpha)$ divides $\boldsymbol{f}_{\boldsymbol{B}}(\alpha)$. Thus, we may assume that $\boldsymbol{B} = \{M, M', H\}$ for some $H \in [\{M, M', \boldsymbol{v}''\vec{\boldsymbol{S}}\}]$, in which case it follows from (60) and (35) that

$$\begin{aligned}\deg_{\boldsymbol{f}_{\{M,M'\}}(\alpha)} \boldsymbol{f}_H(\alpha) &= \deg \boldsymbol{f}_{\boldsymbol{B}}(\alpha) - \deg \boldsymbol{f}_{\{M,M'\}}(\alpha)\\ &< \left(1 + \frac{(q-1)^2}{q^2}\epsilon_q\right) n - \left(1 - \frac{(q-1)^2}{q^2}\epsilon_q\right) n = 2\left(\frac{(q-1)^2}{q^2}\epsilon_q n\right).\end{aligned} \tag{61}$$

We contend that $\boldsymbol{f}_{\boldsymbol{v}''\mathcal{C}(\vec{\boldsymbol{S}})}(\alpha)$ divides $\boldsymbol{f}_{\boldsymbol{B}}(\alpha)$.

Let $\boldsymbol{f}_{\boldsymbol{S}}(\alpha) = \prod_{i=1}^{\ell} p_i^{d_i}(\alpha)$ be the decomposition of $\boldsymbol{f}_{\boldsymbol{S}}(\alpha)$ into irreducible factors. By the definition of $\boldsymbol{f}_{\boldsymbol{v}''\mathcal{C}(\vec{\boldsymbol{S}})}(\alpha)$, it suffices to show that for each $i = 1, 2, \ldots, \ell$, $\boldsymbol{f}_{\boldsymbol{v}''\vec{\boldsymbol{S}}|_i}(\alpha)$ divides $\boldsymbol{f}_{\boldsymbol{B}}(\alpha)$.

Since $\sum_{i=1}^{\ell} \boldsymbol{v}''\vec{\boldsymbol{S}}|_i = \boldsymbol{v}''\vec{\boldsymbol{S}}$ and $H \in [\{M, M', \boldsymbol{v}''\vec{\boldsymbol{S}}\}]$, for each $i = 1, 2, \ldots, \ell$,

$$\boldsymbol{v}''\vec{\boldsymbol{S}}|_i \in [\{M, M', H\} \cup \{\boldsymbol{v}''\vec{\boldsymbol{S}}|_j : j \neq i\}].$$

We observe next that

$$\begin{aligned}\deg \boldsymbol{f}_{\boldsymbol{S}\cup\{H\}}(\alpha) &= \deg \boldsymbol{f}_{\boldsymbol{S}}(\alpha) + (\deg \boldsymbol{f}_{\boldsymbol{S}\cup\{H\}}(\alpha) - \deg \boldsymbol{f}_{\boldsymbol{S}}(\alpha)) \\ &\leq \deg \boldsymbol{f}_{\boldsymbol{S}}(\alpha) + (\deg \boldsymbol{f}_{\boldsymbol{B}}(\alpha) - \deg \boldsymbol{f}_{\{M,M'\}}(\alpha)) \leq 2n,\end{aligned}$$

where the first inequality holds because each factor of $\boldsymbol{f}_H(\alpha)$ that does not divide $\boldsymbol{f}_{\boldsymbol{S}}(\alpha)$ does not divide $\boldsymbol{f}_{\{M,M'\}}(\alpha)$ either, and the second inequality follows from (56) and (61). Therefore,

$$\deg \boldsymbol{f}_{\{M,M',H\}\cup\{\boldsymbol{v}''\vec{\boldsymbol{S}}|_i : i=1,2,\ldots,\ell\}}(\alpha) \leq 2n$$

as well, because $\boldsymbol{f}_{\{M,M',H\}\cup\{\boldsymbol{v}''\vec{\boldsymbol{S}}|_i : i=1,2,\ldots,\ell\}}(\alpha)$ divides $\boldsymbol{f}_{\boldsymbol{S}\cup\{H\}}(\alpha)$. Hence, by Proposition 31, $\boldsymbol{f}_{\boldsymbol{v}''\vec{\boldsymbol{S}}|_i}(\alpha)$ divides $\boldsymbol{f}_{\{M,M',H\}\cup\{\boldsymbol{v}''\vec{\boldsymbol{S}}|_j : j\neq i\}}(\alpha)$.

Consequently, since for all $j = 1, 2, \ldots, i-1, i+1, \ldots, \ell-1, \ell$, $\boldsymbol{f}_{\boldsymbol{v}''\vec{\boldsymbol{S}}|_i}(\alpha)$ and $\boldsymbol{f}_{\boldsymbol{v}''\vec{\boldsymbol{S}}|_j}(\alpha)$ are coprime, $\boldsymbol{f}_{\boldsymbol{v}''\vec{\boldsymbol{S}}|_i}(\alpha)$ divides $\boldsymbol{f}_{\{M,M',H\}}(\alpha)$, and the contention follows.

Therefore,

$$\begin{aligned}\deg \boldsymbol{f}_{\boldsymbol{B}}(\alpha) &= \deg \boldsymbol{f}_{\{M,M',H\}}(\alpha) \\ &\geq \deg \operatorname{lcm}(\{\boldsymbol{f}_{\{M,M'\}}(\alpha), \boldsymbol{f}_{\boldsymbol{v}''\mathcal{C}(\vec{\boldsymbol{S}})}(\alpha)\}) \\ &= \deg \boldsymbol{f}_{\{M,M'\}}(\alpha) + \deg_{\boldsymbol{f}_{\{M,M'\}}(\alpha)} \boldsymbol{f}_{\boldsymbol{v}''\mathcal{C}(\vec{\boldsymbol{S}})}(\alpha) \\ &\geq \deg \boldsymbol{f}_{\{M,M'\}}(\alpha) + \left(1 - \frac{1}{q}\right)(\deg \boldsymbol{f}_{\boldsymbol{S}}(\alpha) - \deg \boldsymbol{f}_{\{M,M'\}}(\alpha)) \\ &\geq \left(1 - \frac{1}{q}\right)(1+\epsilon_q)n + \frac{1}{q}\deg \boldsymbol{f}_{\{M,M'\}}(\alpha) \\ &\geq \left(1 - \frac{1}{q}\right)(1+\epsilon_q)n + \frac{1}{q}\left(1 - \frac{(q-1)^2}{q^2}\epsilon_q\right)n \qquad (62) \\ &\geq \left(1 + \frac{(q-1)^2}{q^2}\epsilon_q\right)n, \qquad (63)\end{aligned}$$

where the first inequality follows from the above contention, the second inequality follows from (57), the third inequality follows from the second clause of Lemma 34, and the fourth inequality follows from (35). That is, we arrived at a contradiction with (60), which completes the proof of the first part of the lemma.

The proof of the second part is similar to the above. The only difference is that, under its assumption, (60) becomes

$$\deg \boldsymbol{f}_{\boldsymbol{B}}(\alpha) < \left(1 + \epsilon_q - \frac{\epsilon_q}{q}\right)n,$$

the second line of (61) becomes

$$\left(1 + \epsilon_q - \frac{\epsilon_q}{q}\right)n - n = \left(\epsilon_q - \frac{\epsilon_q}{q}\right)n,$$

(62) becomes

$$\left(1-\frac{1}{q}\right)(1+\epsilon_q)\,n-\frac{n}{q},$$

and (63) becomes

$$\left(1+\epsilon_q-\frac{\epsilon_q}{q}\right)n.$$

We leave the details to the reader. $\square$

It follows from Proposition 26, by the standard average-counting argument, that we can find a vector $\boldsymbol{v}'' \in F_q^k$ such that

$$\deg_{\boldsymbol{f}_{\{M,M'\}}(\alpha)} \boldsymbol{f}_{\boldsymbol{v}''\mathcal{C}(\vec{\boldsymbol{S}})}(\alpha) \geq \left(1-\frac{1}{q}\right)\deg_{\boldsymbol{f}_{\{M,M'\}}(\alpha)} \boldsymbol{f}_{\boldsymbol{S}}(\alpha). \tag{64}$$

That is, $\boldsymbol{v}''$ satisfies prerequisite (57) of Lemma 39. Therefore, by the first part of the lemma, if

$$\deg \boldsymbol{f}_{\{M,M'\}}(\alpha) < \left(1+\frac{(q-1)^2}{q^2}\epsilon_q\right)n,$$

then

$$\boldsymbol{v}''\vec{\boldsymbol{S}} \notin [\{M,M'\}]. \tag{65}$$

At this point we shall distinguish between the cases of

$$\deg \boldsymbol{f}_{\{M,M'\}}(\alpha) \leq n,$$

i.e., the case of (46),

$$n < \deg \boldsymbol{f}_{\{M,M'\}}(\alpha) < \left(1+\frac{(q-1)^2}{q^2}\epsilon_q\right)n, \tag{66}$$

and

$$\deg \boldsymbol{f}_{\{M,M'\}}(\alpha) \geq \left(1+\frac{(q-1)^2}{q^2}\epsilon_q\right)n.$$

**8.1.1. The case of $\deg \boldsymbol{f}_{\{M,M'\}}(\alpha) \leq n$.** We let $M''$ be $\boldsymbol{v}''\vec{\boldsymbol{S}}$, where $\boldsymbol{v}''$ is as in (64). Then, by the first part of Lemma 39,

$$\deg\{M,M',M''\} \geq \left(1+\frac{(q-1)^2}{q^2}\epsilon_q\right)n$$

and, in view of (65), we just have to show that

$$\|\{M,M',M''\}\| \geq \frac{(q-1)^2}{q^2}\epsilon_q n.$$

For the proof, assume to the contrary that

$$\deg \boldsymbol{f}_H(\alpha) < \frac{(q-1)^2}{q^2}\epsilon_q n$$

for some matrix $H \in [\{M, M', M''\}]$. Then, by (36), $H \notin [\{M, M'\}]$, implying $[\{M, M', H\}] = [\{M, M', M''\}]$. However,

$$\deg \boldsymbol{f}_{\{M,M',H\}}(\alpha) \le \deg \boldsymbol{f}_{\{M,M'\}}(\alpha) + \deg \boldsymbol{f}_H(\alpha) < \left(1 + \frac{(q-1)^2}{q^2}\epsilon_q\right) n$$

contradicts (58).

*Remark* 40. As we have seen in the end of section 7.1, (38) implies (46). In addition, it follows from (4) and (38) that $k \ge 5$. Therefore, the construction of $M, M'$, and $M''$ is completed in the case of (38) as well.

**8.1.2. The case of $n < \deg \boldsymbol{f}_{\{M,M'\}}(\alpha) < (1 + \frac{(q-1)^2}{q^2}\epsilon_q)n$.** Let $\boldsymbol{v}''$ be as in (64). If

$$\|\{M, M', \boldsymbol{v}''\vec{\boldsymbol{S}}\}\| \ge \frac{(q-1)^2}{q^2}\epsilon_q n,$$

then, by Lemma 39 and (65), we let $M''$ be $\boldsymbol{v}''\vec{\boldsymbol{S}}$.

Otherwise, i.e., if

$$\|\{M, M', \boldsymbol{v}''\vec{\boldsymbol{S}}\}\| < \frac{(q-1)^2}{q^2}\epsilon_q n, \tag{67}$$

let $H_{\min} \in [\{M, M', \boldsymbol{v}\vec{\boldsymbol{S}}\}]$ be such that $\deg \boldsymbol{f}_{H_{\min}}(\alpha) = \|\{M, M', \boldsymbol{v}\vec{\boldsymbol{S}}\}\|$. Then, by (67),

$$\deg \boldsymbol{f}_{H_{\min}}(\alpha) < \frac{(q-1)^2}{q^2}\epsilon_q n. \tag{68}$$

Now we need Lemma 41 below.

LEMMA 41. *Let $H$ be a Hankel matrix such that*

$$\deg \boldsymbol{f}_H(\alpha) \le (1 - \epsilon_q)n. \tag{69}$$

*Then, for some $v_H \in F_q$,*

$$\deg\{M, M', H + v_H H_{\min}\} \ge \left(1 + \frac{(q-1)^2}{q^2}\epsilon_q\right) n. \tag{70}$$

We postpone the proof of Lemma 41 until the end of this section and proceed with the construction of $M''$. Let $\boldsymbol{H}$ be as in the beginning of the proof of Lemma 34; see (32).

If for some $H \in \boldsymbol{H} \setminus [\{M, M', H_{\min}\}]$,

$$\|\{M, M', H + v_H H_{\min}\}\| \ge \frac{(q-1)^2}{q^2}\epsilon_q n,[10]$$

we just let $M''$ be $H + v_H H_{\min}$.

Otherwise, i.e., if for each $H \in \boldsymbol{H} \setminus [\{M, M', H_{\min}\}]$,

$$\|\{M, M', H + v_H H_{\min}\}\| < \frac{(q-1)^2}{q^2}\epsilon_q n,$$

[10]Here, of course, $v_H \in F_q$ is from Lemma 41. That is, $v_H$ satisfies (70).

we reduce the case of (66) to Remark 40. The reduction is as follows.

For each $H \in \boldsymbol{H} \setminus [\{M, M', H_{\min}\}]$, let $D_H \in [\{M, M', H + v_H H_{\min}\}]$ be such that

$$\deg \boldsymbol{f}_{D_H}(\alpha) < \frac{(q-1)^2}{q^2}\epsilon_q n. \tag{71}$$

It follows from the definition of $\boldsymbol{H}$ that

$$\dim([\{D_H : H \in \boldsymbol{H} \setminus [\{M, M', H_{\min}\}]\}]) = 2n - o(n).$$

Thus, by Lemma 11, there exists a subset $\boldsymbol{S}'$ of $\{D_H : H \in \boldsymbol{H}\}$ containing $i_q((1+\epsilon_q)n)$ or fewer elements such that $\deg \boldsymbol{f}_{\boldsymbol{S}'}(\alpha) \geq (1+\epsilon_q)n$.

Let

$$\boldsymbol{D}'' = \boldsymbol{D}' \cup \bigcup_{D_H \in \boldsymbol{S}'} \boldsymbol{C}_H,$$

where $\boldsymbol{C}_H$ is as in the proof of Lemma 34. By definition, $\boldsymbol{S}' \subseteq [\boldsymbol{D}'']$. Also

$$\left| \bigcup_{D_H \in \boldsymbol{S}'} \boldsymbol{C}_H \right| \leq \frac{2(1+\epsilon_q)n \lg_q \lg_q n}{\lg_q((1+\epsilon_q)n)}$$

implies

$$|\boldsymbol{D}''| \leq \frac{2(1+\epsilon_q)n \lg_q \lg_q n}{\lg_q((1+\epsilon_q)n)} = o(n).$$

Therefore, we may replace $\boldsymbol{S}$ and $\boldsymbol{D}'$ constructed in Lemma 34 with, respectively, $\boldsymbol{S}'$ and $\boldsymbol{D}''$ that we have just described. Since, by (71),

$$\max\{\deg \boldsymbol{f}_H(\alpha) : H \in \boldsymbol{S}'\} < \frac{(q-1)^2}{q^2}\epsilon_q n,$$

we have the prerequisite of Remark 40.

Thus, to complete the construction, it remains to prove Lemma 41.

*Proof of Lemma* 41. If

$$\deg\{M, M', H\} \geq \left(1 + \frac{(q-1)^2}{q^2}\epsilon_q\right) n,$$

we just let $v_H$ be 0.

Otherwise, i.e., if

$$\deg\{M, M', H\} < \left(1 + \frac{(q-1)^2}{q^2}\epsilon_q\right) n,$$

we proceed as follows.

We observe first that

$$\deg_{\{M,M'\}} \boldsymbol{f}_H(\alpha) < \frac{(q-1)^2}{q^2}\epsilon_q n. \tag{72}$$

Indeed, let $\boldsymbol{B}$ be a basis of $[\{M, M', H\}]$ such that

$$\deg \boldsymbol{f}_{\boldsymbol{B}}(\alpha) < \left(1 + \frac{(q-1)^2}{q^2}\epsilon_q\right) n.$$

Then, by Lemma 33, all $\boldsymbol{f}_M(\alpha), \boldsymbol{f}_{M'}(\alpha)$, and $\boldsymbol{f}_H(\alpha)$ divide $\boldsymbol{f}_{\boldsymbol{B}}(\alpha)$, and (72) follows from the left inequality of (66).

Next, since

$$\deg \boldsymbol{f}_H(\alpha) + \deg \boldsymbol{f}_{H_{\min}}(\alpha) < (1-\epsilon_q)n + \frac{(q-1)^2}{q^2}\epsilon_q n < n,$$

by Remark 24, for all $u, v \in F_q$,

$$\boldsymbol{f}_{(u,v)\mathcal{C}((H,H_{\min})^T)}(\alpha) = \boldsymbol{f}_{uH+vH_{\min}}(\alpha).$$

Therefore, by Proposition 26 with $\boldsymbol{g}(\alpha) = \boldsymbol{f}_{\{M,M'\}}(\alpha)$ and $\boldsymbol{S} = \{H_{\min}, H\}$,

$$\sum_{(u,v)\in F_q^2} \deg_{\boldsymbol{f}_{\{M,M'\}}(\alpha)} \boldsymbol{f}_{uH+vH_{\min}}(\alpha) \geq (q^2-q) \deg_{\boldsymbol{f}_{\{M,M'\}}(\alpha)} \boldsymbol{f}_{\{H_{\min},H\}}(\alpha).$$

Consequently, since

$$\deg_{\boldsymbol{f}_{\{M,M'\}}(\alpha)} \boldsymbol{f}_{H_{\min}}(\alpha) \leq \deg_{\boldsymbol{f}_{\{M,M'\}}(\alpha)} \boldsymbol{f}_{\{H_{\min},H\}}(\alpha),$$

we have

$$\begin{aligned}\sum_{(u,v)\in F_q^2\setminus(\{0\}\times F_q)} & \deg_{\boldsymbol{f}_{\{M,M'\}}(\alpha)} \boldsymbol{f}_{uH+vH_{\min}}(\alpha) \\ & \geq (q^2-2q+1) \deg_{\boldsymbol{f}_{\{M,M'\}}(\alpha)} \boldsymbol{f}_{\{H_{\min},H\}}(\alpha),\end{aligned}$$

and, by the standard average-counting argument, for some $v_H \in F_q$,

$$\begin{aligned}\deg_{\boldsymbol{f}_{\{M,M'\}}(\alpha)} \boldsymbol{f}_{H+v_H H_{\min}}(\alpha) &\geq \left(1-\frac{1}{q}\right) \deg_{\boldsymbol{f}_{\{M,M'\}}(\alpha)} \boldsymbol{f}_{\{H_{\min},H\}}(\alpha) \\ &\geq \left(1-\frac{1}{q}\right) \deg_{\boldsymbol{f}_{\{M,M'\}}(\alpha)} \boldsymbol{f}_{\{H_{\min}\}}(\alpha).\end{aligned}$$

By the definition of $\deg_{\boldsymbol{f}_{\{M,M'\}}}$ (see (16)),

$$\deg_{\boldsymbol{f}_{\{M,M'\}}(\alpha)} \boldsymbol{f}_{H+v_H H_{\min}}(\alpha) = \deg \boldsymbol{f}_{\{M,M',H+v_H H_{\min}\}}(\alpha) - \deg \boldsymbol{f}_{\{M,M'\}}(\alpha)$$

and

$$\deg_{\boldsymbol{f}_{\{M,M'\}}(\alpha)} \boldsymbol{f}_{\{H_{\min}\}}(\alpha) = \deg \boldsymbol{f}_{\{M,M',H_{\min}\}}(\alpha) - \deg \boldsymbol{f}_{\{M,M'\}}(\alpha).$$

Therefore, it follows from the inequality above that

$$\begin{aligned}&\deg_{\boldsymbol{f}_{\{M,M'\}}(\alpha)} \boldsymbol{f}_{\{M,M',H+v_H H_{\min}\}}(\alpha) \\ &\geq \left(1-\frac{1}{q}\right)(\deg \boldsymbol{f}_{\{M,M',H_{\min}\}}(\alpha) - \deg \boldsymbol{f}_{\{M,M'\}}(\alpha)) + \deg \boldsymbol{f}_{\{M,M'\}}(\alpha)\end{aligned}$$

$$
\begin{aligned}
&= \left(1-\frac{1}{q}\right)\deg \boldsymbol{f}_{\{M,M',H_{\min}\}}(\alpha) + \frac{\deg \boldsymbol{f}_{\{M,M'\}}(\alpha)}{q}\\
&\geq \left(1-\frac{1}{q}\right)\left(1+\frac{q-1}{q}\epsilon_q\right)n + \frac{\deg \boldsymbol{f}_{\{M,M'\}}(\alpha)}{q}\\
&= \left(1+\frac{(q-1)^2}{q^2}\epsilon_q\right)n - \frac{n}{q} + \frac{\deg \boldsymbol{f}_{\{M,M'\}}(\alpha)}{q}\\
&\geq \left(1+\frac{(q-1)^2}{q^2}\epsilon_q\right)n,
\end{aligned}
$$

where the second inequality is by the second part of Lemma 39 and the last inequality is by the left inequality of (66).

Now, the proof of (70) is similar to that of Lemma 39. Assume to the contrary that for some basis $\boldsymbol{B}$ of $[\{M, M', H + v_H H_{\min}\}]$,

$$
\deg \boldsymbol{f}_{\boldsymbol{B}}(\alpha) < \left(1+\frac{(q-1)^2}{q^2}\epsilon_q\right)n. \tag{73}
$$

Then it follows from (33), (34), and Lemma 33 that $\boldsymbol{f}_{\{M,M'\}}(\alpha)$ divides $\boldsymbol{f}_{\boldsymbol{B}}(\alpha)$, which allows us to assume that $\boldsymbol{B} = \{M, M', H'\}$ for some $H' \in [\{M, M', H + v_H H_{\min}\}]$.

We have

$$
\begin{aligned}
&\deg \boldsymbol{f}_{\{M,M',H'\}\cup\{H+v_H H_{\min}\}}(\alpha)\\
&= \deg \boldsymbol{f}_{\{M,M',H'\}}(\alpha) + \deg_{\boldsymbol{f}_{\{M,M',H'\}}(\alpha)} \boldsymbol{f}_{H+v_H H_{\min}}(\alpha)\\
&\leq \deg \boldsymbol{f}_{\{M,M',H'\}}(\alpha) + \deg_{\boldsymbol{f}_{\{M,M',H'\}}(\alpha)} \boldsymbol{f}_{H}(\alpha) + \deg_{\boldsymbol{f}_{\{M,M',H'\}}(\alpha)} \boldsymbol{f}_{H_{\min}}(\alpha)\\
&\leq \deg \boldsymbol{f}_{\{M,M',H'\}}(\alpha) + \deg_{\boldsymbol{f}_{\{M,M'\}}(\alpha)} \boldsymbol{f}_{H}(\alpha) + \deg \boldsymbol{f}_{H_{\min}}(\alpha)\\
&\leq \left(1+\frac{(q-1)^2}{q^2}\epsilon_q\right)n + \frac{(q-1)^2}{q^2}\epsilon_q n + \frac{(q-1)^2}{q^2}\epsilon_q n < 2n,
\end{aligned}
$$

where the last inequality follows from (73), (72), and (68).

Thus, by Proposition 31, $\boldsymbol{f}_{\{M,M',H+v_H H_{\min}\}}(\alpha)$ divides $\boldsymbol{f}_{\boldsymbol{B}}(\alpha)$. Since

$$
\deg \boldsymbol{f}_{\{M,M',H+v_H H_{\min}\}}(\alpha) \geq \left(1+\frac{(q-1)^2}{q^2}\epsilon_q\right)n,
$$

this contradicts our assumption (73). □

**8.1.3. The case of $\deg \boldsymbol{f}_{\{M,M'\}}(\alpha) \geq (1+\frac{(q-1)^2}{q^2}\epsilon_q)n$.** This is a particular case of the previous section. The only modification is to ignore Lemma 41 and to replace $H_{\min}$ with $\mathbf{0}_{(n+1)\times n}$. We leave this easy exercise to the reader.

**8.2. The case of $k \leq 4$.** To construct $M''$, we shall treat each of the cases of $k = 2$, $k = 3$, and $k = 4$ separately.

**8.2.1. The case of $k = 2$.** It follows from (54) that $\deg \boldsymbol{f}_{\{M,M'\}}(\alpha) \geq (1+\epsilon_q)n$, which is the case in section 8.1.3.

**8.2.2. The case of $k = 3$.** It follows from (54) that, in particular,

$$\deg \boldsymbol{f}_{\{M,M',H_3\}}(\alpha) \geq \left(1 + \frac{(q-1)^2}{q^2}\epsilon_q\right) n,$$

which is the case in section 8.1.2. The only modification required is to replace $\boldsymbol{v}''\vec{\boldsymbol{S}}$ with $H_3$.

**8.2.3. The case of $k = 4$.** If

$$\deg \boldsymbol{f}_{\{H_1,H_2,H_3\}}(\alpha) \geq \left(1 + \frac{(q-1)^2}{q^2}\epsilon_q\right) n,$$

we are in the case of section 8.2.2.

Otherwise, it follows from (55) and (35) that

$$\deg \boldsymbol{f}_{\boldsymbol{S}}(\alpha) \leq \left(1 + \frac{(q-1)^2}{q^2}\epsilon_q\right) n + 2\left(\frac{(q-1)^2}{q^2}\epsilon_q n\right),$$

which together with (4) implies (56). Therefore, all arguments of section 8.1 apply to the latter case; see Remark 38.

This completes the construction of matrices $M, M'$, and $M''$ and the proof of Theorem 1.

## REFERENCES

[1] A. Averbuch, N. H. Bshouty, and M. Kaminski, *A classification of quadratic algorithms for multiplying polynomials of small degree over finite fields*, J. Algorithms, 13 (1992), pp. 577–588.

[2] R. W. Brockett and D. P. Dobkin, *On the optimal evaluation of a set of bilinear forms*, Linear Algebra Appl., 19 (1978), pp. 207–235.

[3] M. R. Brown and D. P. Dobkin, *An improved lower bound on polynomial multiplication*, IEEE Trans. Comput., 29 (1980), pp. 337–340.

[4] N. H. Bshouty, *A lower bound for the multiplication of polynomials modulo a polynomial*, Inform. Process. Lett., 41 (1992), pp. 321–326.

[5] N. H. Bshouty and M. Kaminski, *Multiplication of polynomials over finite fields*, SIAM J. Comput., 19 (1990), pp. 452–456.

[6] P. Bürgisser, M. Clausen, and A. Shokrollahi, *Algebraic Complexity Theory*, Springer, Berlin, 1997.

[7] D. V. Chudnovsky and G. V. Chudnovsky, *Algebraic complexities and algebraic curves over finite fields*, J. Complexity, 4 (1988), pp. 285–316.

[8] E. Feig, *On systems of bilinear forms whose minimal division-free algorithms are all bilinear*, J. Algorithms, 2 (1981), pp. 261–281.

[9] E. Feig, *Certain systems of bilinear forms whose minimal algorithms are all quadratic*, J. Algorithms, 4 (1983), pp. 137–149.

[10] C. M. Fiduccia and Y. Zalcstein, *Algebras having linear multiplicative complexity*, J. ACM, 24 (1977), pp. 311–331.

[11] J. Ja' Ja', *On the complexity of bilinear forms with commutativity*, SIAM J. Comput., 9 (1980), pp. 713–728.

[12] J. Ja' Ja', *Optimal evaluation of pairs of bilinear forms*, SIAM J. Comput., 8 (1979), pp. 443–462.

[13] J. Ja' Ja', *Computation of bilinear forms over finite fields*, J. ACM, 27 (1980), pp. 822–830.

[14] M. Kaminski, *A lower bound for polynomial multiplication*, Theoret. Comput. Sci., 40 (1985), pp. 319–322.

[15] M. Kaminski and N. H. Bshouty, *Multiplicative complexity of polynomial multiplication over finite fields*, J. ACM, 36 (1989), pp. 150–170.

[16] A. Lempel, G. Seroussi, and S. Winograd, *On the complexity of multiplication in finite fields*, Theoret. Comput. Sci., 22 (1983), pp. 285–296.

[17] A. Lempel and S. Winograd, *A new approach to error-correcting codes*, IEEE Trans. Inform. Theory, 23 (1977), pp. 503–508.
[18] R. Lidl and H. Niederreiter, *Finite Fields*, Encyclopedia Math. Appl. 20, G.-C. Rota, ed., Addison–Wesley, Reading, MA, 1983.
[19] W. W. Peterson and E. Weldon, *Error-Correcting Codes*, MIT Press, Cambridge, MA, 1972.
[20] I. E. Shparlinski, M. A. Tsfasman, and S. G. Vladut, *Curves with many points and multiplication in finite fields*, in Coding Theory and Algebraic Geometry, Lecture Notes in Math. 1518, H. Stichtenoth and M. A. Tsfasman, eds., Springer, Berlin, 1992, pp. 145–169.
[21] V. Strassen, *Vermeidung von divisionen*, J. Reine Angew. Math., 264 (1973), pp. 184–202.
[22] S. Winograd, *On the number of multiplications necessary to compute certain functions*, Comm. Pure Appl. Math., 23 (1970), pp. 165–179.
[23] S. Winograd, *Some bilinear forms whose multiplicative complexity depends on the field constants*, Math. Systems Theory, 10 (1976/77), pp. 169–180.

SIAM J. COMPUT. © 2005 Society for Industrial and Applied Mathematics
Vol. 34, No. 4, pp. 993–1024

# A WORK-OPTIMAL DETERMINISTIC ALGORITHM FOR THE CERTIFIED WRITE-ALL PROBLEM WITH A NONTRIVIAL NUMBER OF ASYNCHRONOUS PROCESSORS[*]

GRZEGORZ MALEWICZ[†]

**Abstract.** Martel [C. Martel, A. Park, and R. Subramonian, *SIAM J. Comput.*, 21 (1992), pp. 1070–1099] posed a question for developing a work-optimal deterministic asynchronous algorithm for the fundamental load-balancing and synchronization problem called Certified Write-All (CWA). In this problem, introduced in a slightly different form by Kanellakis and Shvartsman in a PODC'89 paper [P. C. Kanellakis and A. A. Shvartsman, *Distributed Computing*, 5 (1992), pp. 201–247], $p$ processors must update $n$ memory cells and only then signal the completion of the updates. It is known that solutions to this problem can be used to simulate synchronous parallel programs on asynchronous systems with worst-case guarantees for the overhead of a simulation. Such simulations are interesting because they may increase productivity in parallel computing since synchronous parallel programs are easier to reason about than are asynchronous ones.

This paper presents the first solution to the question of Martel, Park, and Subramonian. Specifically, we show a deterministic asynchronous algorithm for the CWA problem. Our algorithm has the work complexity of $O(n + p^4 \log n)$. This work complexity is asymptotically optimal for a nontrivial number of processors $p \le (n/\log n)^{1/4}$. In contrast, all known deterministic algorithms require superlinear in $n$ work when $p = n^{1/r}$ for any fixed $r \ge 1$.

Our algorithm generalizes the collision principle introduced by Buss et al. [J. Buss, P. C. Kanellakis, P. L. Ragde, and A. A. Shvartsman, *J. Algorithms*, 20 (1996), pp. 45–86] in 1996, which has not been previously generalized despite various attempts. Each processor maintains a collection of intervals of $\{1, 2, \ldots, n\}$. Any processor iteratively selects an interval and works from its tip toward the other tip until it finishes the work or collides with another processor. Collisions are detected effectively using a special Read-Modify-Write operation. In any case, the processor transforms its collection appropriately. Our analysis shows that the transformations preserve some structural properties of collections of intervals. This guarantees that work is assigned to processors in an efficient manner.

**Key words.** design and analysis of parallel algorithms, deterministic algorithms, asynchrony, certified write-all

**AMS subject classifications.** 68Q01, 68W01, 68Q25, 68W10, 68W40

**DOI.** 10.1137/S0097539703428014

## 1. Introduction.

This paper shows a deterministic algorithm where $p$ asynchronous processors update $n$ cells of shared memory and only then signal the completion of the updates. The algorithm has asymptotically optimal work complexity of $O(n)$ for a nontrivial number of processors $p \le (n/\log n)^{1/4}$. This result is the first solution to the question posed by Martel, Park, and Subramonian [33] in 1992.

[*]Received by the editors May 19, 2003; accepted for publication (in revised form) October 4, 2004; published electronically May 27, 2005. An extended abstract of this research appeared in the Proceedings of the 22nd ACM Symposium on Principles of Distributed Computing (PODC'03). The work of Grzegorz Malewicz was done during a visit to the Supercomputing Technologies Group ("the Cilk Group"), Massachusetts Institute of Technology, headed by Prof. Charles E. Leiserson. Grzegorz Malewicz was visiting this group during the 2002/2003 academic year while in his final year of the Ph.D. program at the University of Connecticut where his advisor was Prof. Alex Shvartsman. Part of this research was supported by the Singapore/MIT Alliance.

http://www.siam.org/journals/sicomp/34-4/42801.html

[†]Department of Computer Science, University of Alabama, 116 Houser Hall, Tuscaloosa, AL 35487 and Mathematics and Computer Science Division, Argonne National Laboratory, 9700 South Cass Ave., Argonne, IL 60439 (greg@cs.ua.edu).

Many existing parallel systems are asynchronous. However, writing correct parallel programs on an asynchronous shared memory system is often difficult, for example, because of data races, which are difficult to detect in general [7, 38]. When the instructions of a parallel program are written with the intention of being executed on a system that is synchronous, then it is easier for a programmer to write correct programs because it is easier to reason about synchronous parallel programs than asynchronous ones. Therefore, in order to improve productivity in parallel computing, one could offer programmers the illusion that their programs run on a parallel system that is synchronous, while in fact the programs would be simulated on an asynchronous system.

Simulations of a parallel system that is synchronous on a system that is asynchronous have been studied for over a decade [3, 4, 5, 6, 10, 14, 16, 20, 22, 23, 24, 25, 30, 31, 32, 33, 35, 40, 41]. Simplifying considerably, such simulations assume that there is a system with $p$ asynchronous processors, and that the system must simulate a program written for $n$ synchronous processors. The simulations use three main ideas: idempotence, load balancing, and synchronization. Specifically, the execution of the program is divided into a sequence of phases. A phase executes an instruction for each of the $n$ synchronous programs. A phase is divided into two stages. First, the $n$ instructions are executed and the results are saved to a scratch memory. Only then cells of the scratch memory are copied back to desired cells of the main memory. This ensures that the result of the phase is the same even if multiple processors execute the same instruction in a phase, which may happen due to asynchrony. The $p$ processors run a load-balancing algorithm to ensure that the $n$ instructions of the phase are executed quickly despite possibly varying speeds of the $p$ processors. In addition, the $p$ processors are synchronized at every stage (twice per phase), so as to ensure that the simulated program proceeds in lock-step. Such simulation implements the PRAM model [15] on an asynchronous system.

One challenge in realizing the simulations is the problem of "late writers," i.e., when a slow processor clobbers the memory of a simulation with a value from an old phase. This problem has been addressed in various ways: by replication of variables [23]; by a combination of hashing, replication, and error correction [4]; by approximate detection of who is late, and replication of variables [5]; by using instructions that execute relatively fast [33]; by versioning of variables using extra atomic primitives [34]; or by restricting a class of computations that can be simulated [33].

Another challenge is the development of efficient load-balancing and synchronization algorithms. This challenge is abstracted as the Certified Write-All (CWA) problem. In this problem, introduced in a slightly different form by Kanellakis and Shvartsman [20], there are $p$ processors and an array $w$ with $n$ cells and a flag $f$, all initially 0, and the processors must set the $n$ cells of $w$ to 1, and only then set $f$ to 1. One efficiency criterion for the simulation is to reduce the wasteful use of computing resources. This use can be abstracted as the *work* complexity (or work for short) that is equal to the worst-case total number of instructions executed by the simulation. A simulation uses an algorithm that solves the CWA problem. Therefore, it is desirable to develop low-work algorithms that solve the CWA problem.

When creating a simulation of a given parallel program for $n$ processors, one may have a choice of the number $p$ of simulating processors. On the one hand, when a CWA algorithm for $p \gg n$ is used in a simulation, the simulation may be faster as compared to the simulation that uses an algorithm for $p \ll n$ processors, simply because of higher parallelism, which means that more processors are available to perform the simulation. On the other hand, however, processors that access shared memory may create hotspots, which may cause delays, and as a result an algorithm for $p \gg n$ may in fact run slower than an algorithm for $p \ll n$ (memory contention is disregarded in the model studied in this paper). The actual speed of a simulation may depend on system parameters, and so it is interesting to study CWA algorithms for different relationships between $p$ and $n$.

The best known randomized algorithm that solves the CWA problem on an asynchronous system was given by Martel and Subramonian [36]. Their algorithm has expected work of $O(n)$ when $p \leq n/\log n$, and expected work of $O(n \log n)$ when $p = n$. They also showed a lower bound of $\Omega(n + p \log p)$ on expected work of Las Vegas CWA algorithms against an oblivious adversary.

Deterministic algorithms that solve the CWA problem on an asynchronous system can be used to create simulations that have bounded worst-case overhead. Thus several deterministic algorithms have been studied [1, 8, 9, 18, 21, 37]. Fixing $r \geq 1$ when $p = n^{1/r}$, all these deterministic algorithms have work $\omega(n)$. Specifically, when $r = 1$ the first asynchronous CWA algorithm, called X, was developed by Buss et al. [8]. This algorithm was later generalized by Anderson and Woll [1]. Using a lower bound on *contention* of permutations (a value related to the number of left-to-right maxima in the permutations; see [1] for a formal definition) due to Lovász [29] and Knuth [27], Malewicz [31] showed that the generalized algorithm, called AWT, has work $\Omega(n^{1+\sqrt{\ln\ln n/\ln n}/2})$. When $r \geq 2$, a trivial algorithm, where each processor writes to every cell of the array $w$, has work $\Omega(n^{1+1/r})$. The best known algorithm for $r \geq 2$ was given by Anderon and Woll [1]. This algorithm, called AW, has work $\Omega(n \log n)$, which can be shown using the same lower bound of Lovász and Knuth (this algorithm can be instantiated using results of Naor and Roth [37], Kanellakis and Shvartsman [21], and Chlebus et al. [9], with the same asymptotic lower bound on work). Algorithms X, AW, and AWT access shared memory through atomic read and write instructions. The elegant algorithm of Groote et al. [18] has work $\Omega(n^{1+1/(2r2^r \ln 2)})$, and it uses a Test-and-Set instruction [19]. All these deterministic algorithms have work $\omega(n)$ when $r \geq 1$ is fixed.

An interesting deterministic algorithm, called T, for 3 processors was shown by Buss et al. [8]. In this algorithm, two processors start from the two opposite tips of the array $w$, and each works toward the opposite tip. The third processor starts from the middle of the array and "expands" by setting to 1 further and further cells on each side of the starting cell. When a "collision" between two processors occurs, the two processors "jump" to repeat the pattern of work recursively in a different part of the array. The algorithm has work $O(n)$, and at most $n + O(\log n)$ cells of the array are set to 1. A generalization of this algorithm to more than 3 processors is an open problem posed by Buss et al. in 1996. In a recent paper by Groote et al. [18] it is stated that "Algorithm T does not appear to be generalizable to larger numbers of processes."

**Contributions.** This paper presents, for the first time, a deterministic algorithm for the CWA problem that has asymptotically optimal work for a nontrivial number of asynchronous processors. Specifically, we consider a shared memory setting with $p$ asynchronous processors. The processors must solve the CWA problem: given an array $w$ with $n$ cells and a flag $f$, all initially 0, set all elements of $w$ to 1, and only then set $f$ to 1. We present a deterministic algorithm that solves the problem in this shared memory setting. Our algorithm has the work complexity of $O(n + p^4 \log n)$ (appearing in Theorem 3.15). The algorithm has asymptotically optimal work $O(n)$ for a nontrivial number of processors $p \leq (n/\log n)^{1/4}$. In contrast, all known deterministic algorithms require as much as $\omega(n)$ work when $p = n^{1/r}$, for any fixed $r \geq 1$. The processors use $O(n + p^4 \log n)$ memory cells for coordinating their work (shown in Theorem 3.16). Our algorithm generalizes the collision principle used by the algorithm T. Namely, each processor has a collection of intervals of $w$ and iteratively selects an interval to work on. The processor proceeds from one tip of the interval toward the other tip. When processors collide, they exchange appropriate information and schedule their future work accordingly. Our algorithm uses a special atomic Read-Modify-Write (RMW) instruction to detect collisions. Such strong primitives were not used by previous algorithms, except for the algorithm of Groote et al. [18]. Our paper contributes to solving the problems posed by Martel, Park, and Subramonian [33] and by Buss et al. [8].

**Subsequent work.** Subsequent to the conference version of this paper [30], Kowalski and Shvartsman presented [26] a deterministic asynchronous algorithm for the CWA problem. Their algorithm has asymptotically optimal work when the number of processors is $p < n^{1/(2+\epsilon)}$. This range is significantly wider than the range of $p$ while the algorithm presented here is proven to have asymptotically optimal work. However, it is not clear that our upper bound on work is tight. It would be interesting to develop tight bounds on work for each algorithm and compare the bounds. The algorithm of Kowalski and Shvartsman uses atomic reads and writes, while our algorithm requires a much stronger primitive of RMW. Their algorithm uses a collection of $q$ permutations with contention $O(q \log q)$, while it is not known to date how to construct such permutations in polynomial time. Thus their result is, so far, existential, while ours is explicit.

**Paper organization.** The remainder of the paper is organized as follows. In section 2, we define the asynchronous shared memory model of computation used in the paper and the CWA problem. In section 3, we present our deterministic algorithm and its analysis. Finally, in section 4, we conclude with future work.

## 2. Model and definitions.

We consider a shared memory system where processors can work at arbitrarily varying paces. Our formal definition is based on the Atomic Asynchronous Parallel System as presented by [5] (cf. [2, 11, 12, 13, 17, 28, 33, 39, 42]).

The system consists of $p$ processors, each of which has a dedicated local memory, and every processor has access to shared memory. Any memory is composed of *cells*. The initial section of $n$ cells of shared memory stores an array $w[0, \ldots, n-1]$. The subsequent cell stores a flag $f$. Any cell of any memory can store any $O(\log n)$-bit number. Any processor has a distinct identifier from $\{1, \ldots, p\}$.

Each processor has a discrete local clock ranging over $\mathbb{N} = \{1, 2, 3, \ldots\}$. A processor executes exactly one *basic action* at any tick of the local clock unless the processor has halted. The basic actions that a processor can execute are a $Halt$ action that stops the operation of the processor, any operation on a constant number of cells from the local memory, and a transfer between the local memory and shared memory. The possible transfers are reading a single cell of shared memory into a cell of the local memory; writing from a cell of the local memory to a cell of shared memory; and performing a special RMW action that compares the value stored at a cell of shared memory with the value stored at a cell of the local memory, and if they are equal, the action transfers a constant number of cells from the local memory to a constant number of cells of shared memory, but in any case returns the result of the comparison

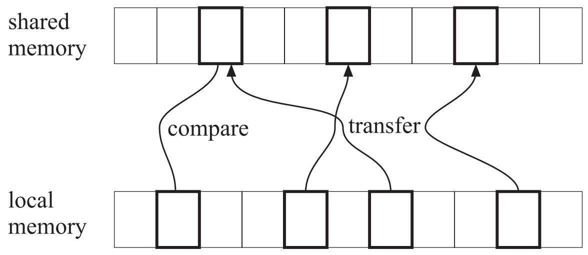


FIG. 1. *The special RMW operation used by the algorithm first compares two cells, and then possibly transfers a constant number of cells from local memory to shared memory. All this is done atomically.*

(see Figure 1 and also an example of syntax in Figure 2).

An execution of an algorithm progresses according to the following model of asynchrony. Local time of processor $i$ is mapped to global time through a strictly increasing function $T_i : \mathbb{N} \to \mathbb{R}$. We assume that no local clock ticks of two processors are mapped to the same instant of global time, i.e., if $T_i(x) = T_j(y)$, then $i = j$ and $x = y$. A tuple $\langle T_1, \ldots, T_p \rangle$ with mappings that satisfy these conditions is called *a valid tuple of mappings*. When a valid tuple $\langle T_1, \ldots, T_p \rangle$ has been fixed, each processor executes basic actions dictated by its algorithm. The processors take turns according to the total order prescribed by the mappings. Any processor $i$ does not execute basic actions after the tick when the processor executed the $Halt$ action, if the processor executed the action. The execution of any basic action is instantaneous, and so the resulting memory updates are atomic.

We adopt the following definition of the CWA problem: given the array $w[0, \ldots, n-1]$ with $n$ cells and the flag $f$, all initially 0, set the $n$ cells of $w$ to 1, and only then set $f$ to 1. An algorithm *solves* the CWA problem for $p$ processors and $n$ cells if, for any valid tuple $\langle T_1, \ldots, T_p \rangle$ of mappings, the following three conditions hold:

(i) (Termination) each processor halts after a finite number of local clock ticks,
(ii) (Certification) when any processor halts, the flag $f$ has been set to 1,
(iii) (Validity) when the flag $f$ is set to 1, all cells of $w$ have been set to 1.

The *work* complexity of a deterministic algorithm that solves the CWA problem for $p$ processors and $n$ cells measures the maximum total number of basic actions executed by the processors. Consider any valid tuple $\langle T_1, \ldots, T_p \rangle$ of mappings. Let $h_i$ be the first local clock tick when processor $i$ executes the $Halt$ action, or $\infty$ if it does not execute the action. Then the total number of basic actions executed by the processors is $\sum_{i=1}^{p} h_i$. The work of the algorithm is defined as the maximum value of the sum across valid tuples of mappings. (Work is a function of $n$ and $p$.)

DEFINITION 2.1. *The work of a deterministic algorithm $A$ for $p$ processors and $n$ cells that solves the CWA problem is defined as*

$$work(A, p, n) = \max_{\langle T_1, \ldots, T_p \rangle} \sum_{i=1}^{p} h_i(T_1, \ldots, T_p),$$

*where the maximum is taken over all valid tuples of mappings; and where $h_i(T_1, \ldots, T_p)$ is a number from $\mathbb{N}$ that is the first tick of the local clock of processor $i$ during which the processor executes the Halt basic action, when local time of processors have been mapped to global time using the maps $T_1, \ldots, T_p$.*

Note that in this model, there is a trivial Write-All algorithm for $n = p$ where the first basic action that a processor $i$, $0 \leq i \leq n-1$, executes is an assignment of 1 to cell $i$ of the array $w$ (because the model ensures that each processor will eventually perform a basic action). This takes $O(n)$ work in total. However, in general, no processor can certify and halt immediately after performing its first basic action without violating the validity condition. The processor simply cannot always ensure that each of the $n$ cells has been set to 1, due to the fact that other processors may be delayed.

## 3. Collision algorithm and its analysis.

This section presents a deterministic algorithm for the CWA problem with asynchronous processors (see Figure 2). The algorithm generalizes the collision principle of algorithm T. The main algorithmic approaches of our algorithm are: to ensure that any processor often works on a relatively large interval of unset cells of the array $w$, according to a sequence that enables rapid detection of two processors setting to 1 the same cell of the array; and, when redundancy occurs, to ensure an effective mechanism for reassigning work to processors. Briefly speaking, all processors share an array *tab* with $n$ cells used for coordination of their work. Each processor maintains a collection of intervals of the set $\{0, 1, \dots, n-1\}$ (an interval is a subset of consecutive elements of the set). A processor takes an interval from the collection and keeps setting cell $w[x]$ to 1 and storing some special information in cell $tab[x]$, while working through cells $x$ from a tip of this interval toward the opposite tip. Later, the processor removes some intervals or their parts from the collection, possibly based on information obtained from other processors. This process is repeated as long as there is an interval in the collection. When the collection becomes empty, then the processor sets the flag $f$ to 1 and halts.

There are several challenges that we solve to ensure that our algorithm avoids doing too much redundant work. It may happen that two processors "collide" at the same cell while working in opposite or the same directions. When they work in opposite directions, they could "cross" each other and duplicate the work that the other processor already did. When they work in the same direction, they may keep on working "side-by-side" and again duplicate the work that the other processor is doing. Another potential problem is that even if we are able to detect collisions, then a processor that collides must decide upon a cell of the array where the processor will resume its work. Ideally, the processor should choose to work from a tip of an interval so that this tip is "far away" from any cell that any other processor is currently working on. This is desirable because it would help to ensure that when the next collision of this processor occurs, a substantial number of distinct new cells of the array $w$ have been set to 1.

Intuitively, our algorithm solves these challenges as follows. The processors coordinate their work on intervals using atomic RMW instructions. This ensures that whenever a processor does a successful RMW to a cell, no other processor can succeed. As a result, a colliding processor sets at most one cell of $w$ to 1 before it detects a collision with another processor, and has an opportunity to reassign its own future work. The choice of a relatively long interval located in a rather unassigned part of the array is intuitively done by a processor always working on an interval that is at least as long as half of the length of a longest interval in the collection. In addition, we ensure that a colliding processor obtains knowledge from the other processor about which cells of $w$ remain to be set to 1, and this allows us to guarantee that when a processor often collides, it must substantially reduce the amount of work that it "thinks" remains to be done, even though it has *not* actually recently set to 1 any distinct cells of $w$.

The following sections present details of this intuitive explanation. We begin, in

```
shared variables: $f$, arrays $w[0,\dots,n-1]$, $tab[0,\dots,n-1]$
initially $f=0$, $w[x]=0$, $tab[x]=\langle 0,\emptyset,\emptyset\rangle$, for $x=0,\dots,n-1$

COLLIDE
local variables: $dir, U, D, dir', U', D', c, s, e, x, failed$
01 $U := \{[0,n-1]\}$, tips are unmarked unless explicitly marked
02 while true
--------------------------------------------------------------------------------
03     let $s$ be an unmarked tip of an interval from $U$ and $e$ the other tip  } select an interval
04     if $s \le e$, then $dir := R$ else $dir := L$                             }
--------------------------------------------------------------------------------
05     $D := \emptyset$                                                          } work from tip $s$ to tip
06     for $x := s$ to/downto $e$                                                } $e$ of the interval
07         $w[x] := 1$                                                           }
08         RMW($x$)            // see line 30                                    }
09         if $failed$, then goto 11                                             }
10         $D := D \cup \{x\}$                                                   }
--------------------------------------------------------------------------------
11     if not $failed$, then                                                     } no collision,
12         $U := U \setminus D$                                                  } record progress
--------------------------------------------------------------------------------
13     else                                                                      } collision,
14         $\langle dir', U', D'\rangle := tab[x]$                               } retrieve information
--------------------------------------------------------------------------------
15         if $maxlen(U) > maxlen(U')$, then $U := U' \cap U$                    } intersect collections
16         else                                                                  }
17             $U := U \cap U'$                                                  }
--------------------------------------------------------------------------------
18         if $dir \ne dir'$, then                                               } head-on-head collision,
19             $U := U \setminus (D \cup D')$                                    } record joint progress
--------------------------------------------------------------------------------
20         else                                                                  } head-on-back collision,
21             if $D \ne \emptyset$, then $U := U \setminus D$                   } record progress,
22             else $U := U \setminus (D' \setminus \{x\})$                      } and mark a tip
23             mark tip $x$ of the interval in $U$ that contains $x$             }
--------------------------------------------------------------------------------
24     remove from $U$ any interval of length 1 with a marked tip                } check for completion
25     if $U$ is empty, then set $f$ to 1 and Halt                               }
--------------------------------------------------------------------------------
26     if all tips of all intervals in $U$ are marked, then                      } all tips marked,
27         let $[a,b]$ be an interval among the longest ones in $U$              } split an interval
28         $c := a + \frac{b-a+1}{2} - 1$                                        }
29         $U := (U \setminus \{[a,b]\}) \cup \{[a,c],[c+1,b]\}$                 }
--------------------------------------------------------------------------------

RMW($x$)
30 begin atomic
31     if $tab[x].D$ is $\emptyset$, then
32         $failed := false$
33         $tab[x] := \langle dir, U, D \cup \{x\}\rangle$
34     else $failed := true$
35 end atomic
```

FIG. 2. *Deterministic collision algorithm as executed by any processor; see section* 3.1. *Operations from lines* 15 *to* 17*,* 19*,* 21*, and* 22 *are detailed in section* 3.2*. Theorem* 3.16 *explains how to modify the code so that RMW transfers at most a constant number of memory cells.*

section 3.1 with an overview of the collision algorithm to help the reader gain familiarity with the code of the algorithm and how the algorithm works. In the algorithm, processors maintain collections of intervals and sometimes incorporate knowledge from collections of other processors. Then in section 3.2, we detail the operations on collections of intervals that processors use. The operations incorporate knowledge so that the resulting collections of intervals are "well-behaved" in a precise sense. Finally, in section 3.3, this good behavior of operations helps us prove a bound on work for our algorithm.

**3.1. Overview of the collision algorithm.** We outline the collision algorithm. We begin with the shared and the local memory variables that processors access, and then describe how the algorithm works. Recall that $n$ denotes the number of cells in the Write-All array and $p$ denotes the number of processors. We assume that $n$ and $p$ are powers of two. Line numbers mentioned in this section refer to the code in Figure 2.

Each processor has access to three shared variables: the completion flag $f$, the Write-All array $w[0,\ldots,n-1]$, and an array $tab[0,\ldots,n-1]$ called the *trace table*. The trace table is of paramount significance to the collision algorithm because information about completed work is disseminated through the trace table. Each cell $tab[x]$ of the trace table is a record with three fields denoted by $tab[x].dir$, $tab[x].U$, and $tab[x].D$ (an example of content of a trace table is given in Figure 3(a)). The first field is a bit equal to either $L$ or $R$. The second field is a collection of intervals of $\{0,\ldots,n-1\}$. As we will see later, at most $p/2$ intervals are ever stored in this field. The third field is an interval of $\{0,\ldots,n-1\}$. Note that each interval can be represented as two numbers—the interval tips. Hence, any cell of $tab$ contains $O(p \log n)$ bits. We will later see how to use a pointer representation of collections so as to ensure that each cell of the trace table stores only $O(\log n)$ bits. For now, we use the "expanded" representation for simplicity of exposition. We assume that when processors begin execution, these shared variables are initialized as follows: $f$ and cells of $w$ to zero, and every cell of $tab$ to $\langle L,\emptyset,\emptyset\rangle$ (i.e., $tab[x].dir = L$, $tab[x].U = \emptyset$, and $tab[x].D = \emptyset$, for any $0 \le x \le n-1$).

Each processor has a few local variables. The variable $U$ contains a collection of intervals of $\{0,\ldots,n-1\}$. As we will see later, $U$ can contain at most $p/2$ intervals at any time during execution. These intervals cover all cells that remain to be written to, and so any cell that is not in one of the intervals must necessarily have already been written to. However, some cells covered by intervals from $U$ may already be written to, because of the concurrent work of other processors and possibly delayed dissemination of knowledge. Each tip of any interval in $U$ has a flag equal to *marked* or *unmarked*. The processor has other local variables: $U'$ is a collection of intervals of $\{0,\ldots,n-1\}$ (again, it will contain at most $p/2$ intervals); $D$ and $D'$ are intervals of $\{0,\ldots,n-1\}$; $c$, $s$, $e$, $x$ are integers from $\{0,\ldots,n-1\}$; $dir$, $dir'$, and $failed$ are bits.

Note that the algorithm is uniform (i.e., it is the same for each processor), so we describe it for a given processor $i$. The processor begins by setting its collection $U$ to a set that contains just one interval $[0,n-1]$ (line `01`). The tips of this interval are unmarked. Note that then $U$ contains an interval with an unmarked tip. Next the processor enters a big while loop (lines `02` to `29`). In general, every time the processor starts an iteration of the while loop (line `03`), the following loop invariant holds: the collection $U$ contains at least one interval with an unmarked tip; the other tip and some tips of other intervals, if any, may be marked. The body of the while loop has several sections of code, each carrying out a specific function.

The processor selects an interval from $U$ (lines `03` and `04`). The interval is chosen so that it has an unmarked tip $s$ and $e$ is the other tip of the interval. The processor will attempt to work on the interval from $s$ to $e$. The relationship between $s$ and $e$ determines the direction of work: either $L$, meaning left, or $R$, right.

Then the processor works on the interval by iterating through cells $x$ from tip $s$ to tip $e$ (lines `05` to `10`). At the end of each iteration, the interval $D$ contains the

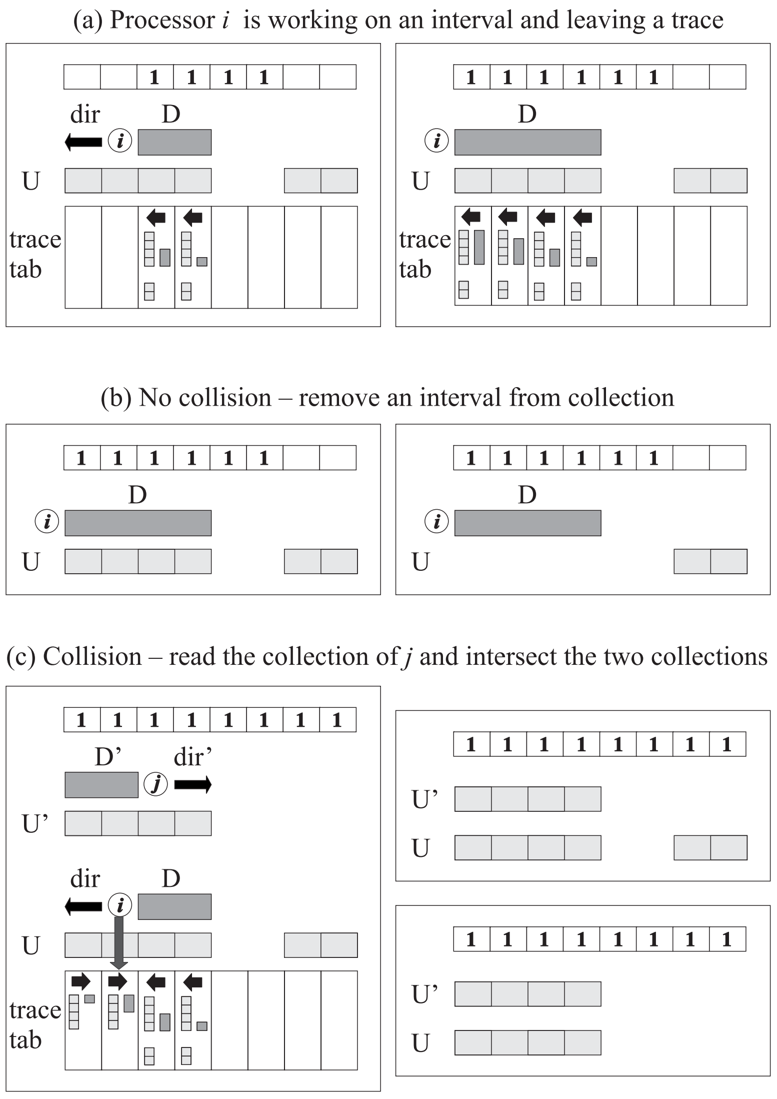

(a) Processor i is working on an interval and leaving a trace
1 1 1 1
dir
D
i
U
trace tab
1 1 1 1 1 1
D
i
U
trace tab
(b) No collision – remove an interval from collection
1 1 1 1 1 1
D
i
U
1 1 1 1 1 1
D
i
U
(c) Collision – read the collection of j and intersect the two collections
1 1 1 1 1 1 1 1
D'
dir'
j
U'
dir
D
i
U
trace tab
1 1 1 1 1 1 1 1
U'
U
1 1 1 1 1 1 1 1
U'
U

Fig. 3. *Illustration of activities of the collision algorithm. Part* (a) *shows a processor that "leaves a trace" in the trace table. Part* (b) *presents changes to U when there has been no collision. Part* (c) *demonstrates the process of retrieval of information about the colliding processor and intersection of the two collections of intervals.*

cells through which the processor has iterated so far. At every iteration of the loop, the processor writes to the cell $w[x]$ of the Write-All array, and "leaves a trace" of its work inside a cell $tab[x]$ of the trace table (see animation in Figure 3(a)). Writing to cell $tab[x]$ is done using the RMW atomic operation so as to ensure that no two processors succeed in writing to a cell of $tab$ (lines 30 to 35). Specifically, the write tests the value of the third field of the cell, and only if it is equal to the empty set, the write proceeds by setting the field to a value of $D \cup \{x\}$ that is always different from the empty set. The interval $D \cup \{x\}$ contains all cells to which the processor has successfully RMWed during the work on the interval. In addition to modifying the third field, a successfully performed RMW stores in $tab[x]$ the direction $dir$ of work and the collection $U$. Two events can happen during the time when the processor iterates from $s$ to $e$: (1) either the processor reaches the tip $e$ of the interval because it has successfully RMWed to all cells of the interval or (2) the processor fails on a RMW operation because earlier some processor $j$ successfully RMWed to a cell of the interval; if this happens we say that processor $i$ *collides* with processor $j$. The actions of processor $i$ depend on whether $i$ collided or not.

When processor $i$ has not collided, it records recent progress (lines `11` and `12`). Specifically, then the interval $D$ is equal to the interval from $U$ with tips $s$ and $e$ on which the processor has just finished working. The processor removes this interval from $U$ (see Figure 3(b)).

Slightly more complicated operations are performed when processor $i$ has collided. In such a case, processor $i$ will incorporate the knowledge gained from the processor $j$ with whom the collision has occurred. The incorporation of knowledge proceeds in several stages.

First, processor $i$ retrieves information about processor $j$ from the trace table (lines `13` and `14`). Recall that a collision occurs when a processor fails on an RMW to a cell $tab[x]$ of the trace table. This means that a processor $j$ has already performed a successful RMW to the cell. This cell must contain the collection $U$ of intervals that processor $j$ had at the time when it performed a successful RMW to the cell $tab[x]$. The cell also contains the direction $dir$ in which processor $j$ was working at that time, and the part $D$ of the interval that processor $j$ had successfully worked on until the time it performed the RMW. So processor $i$ can retrieve these three pieces of information from the trace table by reading $tab[x]$ (see Figure 3(c)). We denote the pieces by $U'$, $dir'$, and $D'$. Note that the local variables $U$, $dir$, and $D$ of processor $j$ at the time when $i$ retrieves information from the cell $tab[x]$ may be different from $U'$, $dir'$, and $D'$ because $j$ might have executed many instructions since it performed RMW to $tab[x]$.

Second, processor $i$ intersects the two collections of intervals, its own $U$ with the collection $U'$ of processor $j$ (lines `15` and `17`). A collection with the longest interval is taken, and its intervals are trimmed by the union of the intervals from the other collection. A detailed specification of the intersection operation is given in Lemma 3.3. As a result of the intersection, $U$ contains some intervals.

Then the actions that processor $i$ takes depend on whether the colliding processors $i$ and $j$ worked in the same or opposite directions.

If they worked in opposite directions, then processor $i$ records progress $D$ and $D'$ (lines `18` and `19`). It removes from $U$ all intervals, or their parts, that are contained in either the interval $D$ that $i$ successfully worked on or the interval $D'$ that $j$ successfully worked on (see Figure 4(a)). A detailed specification of the removal operation is given in Lemma 3.5.

If, on the other hand, processors $i$ and $j$ worked in the same direction, then progress $D$ and $D'$ is recorded in a different way (lines `20` to `23`). Two kinds of a "head-on-back" collision are distinguished. The first kind is when processor $i$ failed on its first RMW in the work on the interval from $s$ to $e$, and the second kind is when processor $i$ succeeded on the first RMW and thus "bumped into the back" of the trace of processor $j$ later during the work on the interval. When processor $i$ succeeded on the first RMW, then $D \neq \emptyset$ (line `21`). Here processor $j$ cannot have arrived at $x$ "from behind" of processor $i$, and so $j$ must have just started working on its interval when it RMWed to cell $tab[x]$ (see Figure 4(b)). Then processor $i$ removes from $U$ the

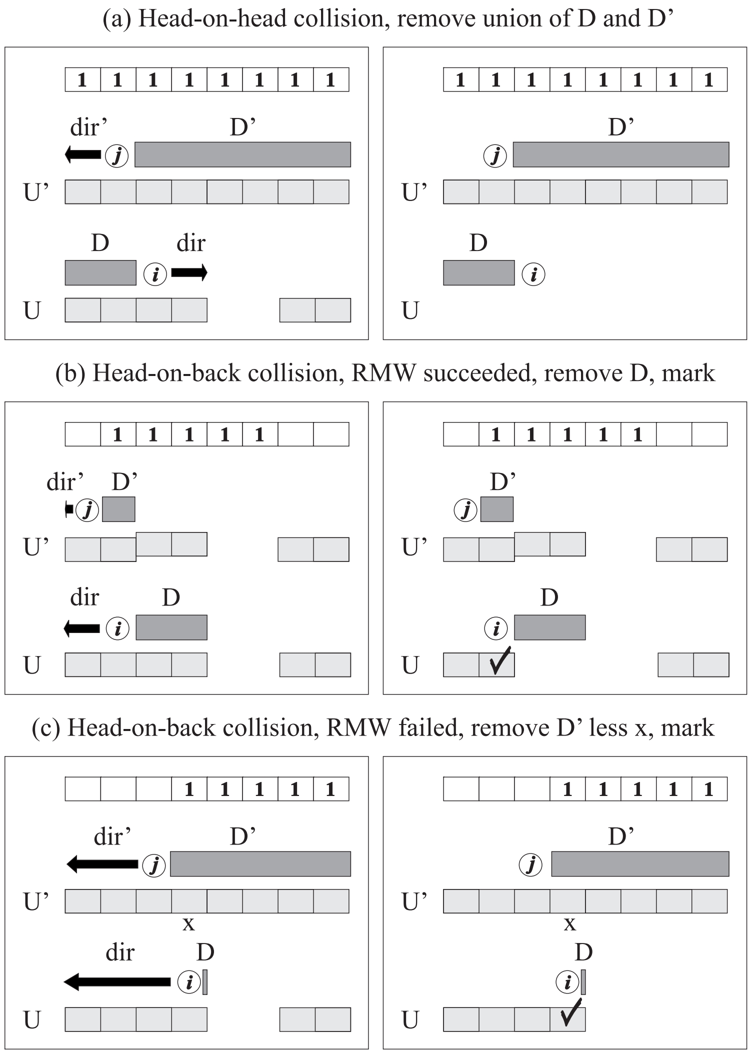

(a) Head-on-head collision, remove union of D and D'
1 1 1 1 1 1 1 1
dir'
D'
j
U'
D
dir
i
U
(b) Head-on-back collision, RMW succeeded, remove D, mark
1 1 1 1 1
(c) Head-on-back collision, RMW failed, remove D' less x, mark
x

FIG. 4. *Illustration of activities of the collision algorithm. Parts* (a), (b), *and* (c) *represent modifications to collection $U$ using intervals $D$, $D'$, and the cell $x$ where collision took place.*

interval $D$ that it has just successfully worked on (line 21). A detailed specification of the removal operation is given in Lemma 3.7. As a result of the removal, cell $x$ becomes a tip of an interval in $U$. When, on the other hand, processor $i$ failed on the first RMW, then $D = \emptyset$ (line 22). Here processor $j$ must have successfully worked on at least one cell, but possibly more (see Figure 4(c)). Processor $i$ removes from $U$ the interval $D'$ that $j$ worked on, except for the cell $x$ where collision took place (line 22). A detailed specification of the removal operation is given in Lemma 3.9. No matter which of the two kinds of "head-on-back" collisions occurred, as a result $x$ is a tip of an interval in $U$, and processor $i$ marks the tip (line 23). Marking allows us to keep track of the tips where head-on-back collisions took place. A marked cell is known to have been set to 1.

In any case, whether processor $i$ collided or not, the processor then checks if all cells have been written to (lines 24 and 25). As the algorithm iterates, intervals of length 1 with a marked tip may emerge in the collection $U$. Such intervals are removed from $U$, as we are certain that the corresponding cells must have been written to. If $U$ does not contain any more intervals, then the processor certifies and halts. As a result of the removal, any interval with a marked tip has length at least 2.

The final action performed by processor $i$ in the body of the while loop is a possible partition of an interval (lines 26 to 29). If all tips of all intervals in $U$ are marked, then the processor takes a longest interval in $U$ and partitions it into two halves. The two tips that are being exposed are unmarked. This ensures that there is at least one interval in $U$ with an unmarked tip. So the while loop invariant holds again.

**3.2. Collections of intervals, their transformations, and preserved properties.** During the time when a processor executes the collision algorithm, the processor interacts with other processors through the trace table. As a result, the processor transforms its collection $U$ in various ways. Lines 11 through 29 of the algorithm list the conditions under which the transformations are performed. The current section is devoted to defining these transformations and demonstrating their effect on $U$. We first introduce the notions of "regularity" and "monotonicity" of collections of intervals. We then show that the transformations maintain regularity and monotonicity of $U$, and that the number of elements contained in the intervals of $U$ gets reduced rapidly under certain conditions. These observations will be useful when reasoning about correctness and work of the algorithm.

We define regularity and monotonicity of collections of intervals (see Figure 5 for illustration). All intervals in this and subsequent sections are over the set of integers $\{0, 1, \ldots, n-1\}$.

DEFINITION 3.1. *Let $U_0, \ldots, U_g$ be collections of intervals over $\{0, \ldots, n-1\}$. We say that the collections are* regular *iff the following three properties are satisfied:*

(i) *Each collection is composed of mutually disjoint nonempty intervals, each interval has length that is a power of two (length can be 1), and the lengths of any two intervals from the same collection differ by at most a factor of 2.*

(ii) *For any two intervals $I$ and $J$, each from a different collection, either $I \subseteq J$, or $J \subseteq I$, or $I \cap J = \emptyset$.*

(iii) *If $I \subseteq J$ are two intervals from different collections, then $J$ can be partitioned into a number of intervals such that this number is a power of 2 (the number can be 1), the intervals have the same length, and $I$ is one of the intervals.*

*We say that the collections are* monotonic *iff the following property is satisfied:*

(iv) *For any $j$, $0 \leq j < g$, if interval $I$ belongs to $U_{j+1}$, then there is interval $J$ that belongs to $U_j$ such that $J \supseteq I$.*

We now show a simple fact that halving any interval in the last collection preserves monotonicity.

LEMMA 3.1. *Let $U_0, \ldots, U_g$ be monotonic collections of intervals, and let $U$ be equal to $U_g$ except that instead of an interval from $U_g$ of length 2 or more, $U$ contains the two halves of the interval. Then the collections $U_0, \ldots, U_g, U$ are monotonic.*

The following lemma states that halving a relatively long interval in any collection preserves regularity.

LEMMA 3.2. *Let $U_0, \ldots, U_g$ be regular collections of intervals, $0 \leq i \leq g$, and let $U$ be equal to $U_i$ except that instead of a longest interval from $U_i$ of length 2 or more, $U$ contains the two halves of the interval. Then the collections $U_0, \ldots, U_g, U$ are regular.*

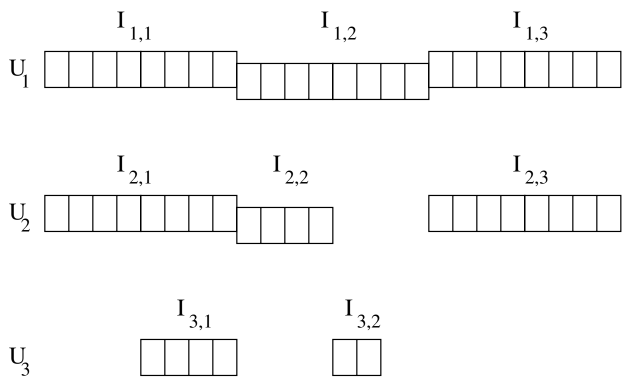


FIG. 5. *This figure illustrates the definition of regular and monotonic collections of intervals. In the example above we have three collections $U_1$, $U_2$, and $U_3$ of intervals of cells. These collections are regular for the following reasons. Any collection has disjoint intervals whose lengths are powers of 2, and the lengths of intervals in any collection differ by the factor of 2, at most (they do not differ for $U_1$, but differ for $U_2$ and for $U_3$). Any two intervals from any two distinct collections are either disjoint or one is contained in the other, i.e., there are no partial overlaps. If one is contained in the other, then the subset must be properly aligned inside the superset, i.e., the superset can be partitioned into intervals of the same length, the number of these intervals is a power of two, and the subset is one of the intervals (e.g., $I_{2,1}$ can be partitioned into two intervals, the right of which is $I_{3,1}$). Note that collections $U_1, U_2$ are monotonic, because for each interval of $U_2$ there is a superset interval in $U_1$. On the other hand, collections $U_2, U_3$ are not monotonic because there is no interval in $U_2$ that contains interval $I_{3,2}$.*

Taking a specific intersection of two collections preserves regularity and monotonicity, as shown below. In this intersection, a collection with the longest interval is taken, and its intervals are trimmed by the union of the intervals from the other collection.

LEMMA 3.3. *Let $k_1, \dots, k_p \ge 0$, let the collections $U_1^0, \dots, U_1^{k_1}, \dots\dots, U_p^0, \dots, U_p^{k_p}$ be regular, and let the collections $U_h^0, \dots, U_h^{k_h}$ be monotonic for any $h$, $1 \le h \le p$. Let $i \ne j$ be two numbers from $\{1, \dots, p\}$, $0 \le m \le k_j$, and $U = U_i^{k_i}$ and $U' = U_j^m$. Let $2^k$ be the maximum length of an interval in $U$ and $2^{k'}$ be the maximum length of an interval in $U'$. Let $V$ be the collection*

$$V = \begin{cases} U' \cap U := \left\{ \ H \mid H \ne \emptyset \ \wedge \ J \in U' \ \wedge \ H = J \cap \bigcup_{I \in U} I \ \right\} & \text{if } k > k', \\ U \cap U' := \left\{ \ H \mid H \ne \emptyset \ \wedge \ I \in U \ \wedge \ H = I \cap \bigcup_{J \in U'} J \ \right\} & \text{if } k \le k'. \end{cases}$$

*Then the number of intervals in $V$ is at most the maximum of the number of intervals in $U$ and in $U'$, i.e., $|V| \le \max\{|U|, |U'|\}$. If $k > k'$, then $V$ contains some complete intervals from $U'$, and when $k < k'$, some complete intervals from $U$. If $k = k'$, then $V$ contains some complete intervals from $U$ and a single half for each of some other intervals of length $2^k$ from $U$ ($V$ never contains two halves of any interval from $U$). The collections $V, U_1^0, \dots, U_1^{k_1}, \dots, \dots, \dots, U_p^0, \dots, U_p^{k_p}$ are regular, and the collections $U_i^0, \dots, U_i^{k_i}, V$ are monotonic.*

*Proof.* We shall study the content of $V$ based on the relationship between the length of a longest interval in $U$ and the length of a longest interval in $U'$. In preparation for the case analysis we record what the lengths of intervals in these collections may be. Since collections are regular, by property (i), we indeed know that the length of a longest interval in $U$ is $2^k$ and the length of a longest interval in $U'$ is $2^{k'}$, for some integers $k, k' \geq 0$. We also know that intervals in $U$ have length $2^k$ or $2^{k-1}$, and the intervals in $U'$ have length $2^{k'}$ or $2^{k'-1}$.

For the first case, suppose that $k > k'$, and let us investigate common parts between $U'$ and $U$. Let $I$ be an interval from $U$, and $J$ from $U'$. The interval $I$ cannot be shorter than $2^{k-1}$, and the interval $J$ cannot be longer than $2^{k-1}$. Therefore, $J$ is too short to be a strict superset of $I$. Hence, by property (ii), any interval $J$ from $U'$ is either a subset of some interval from $U$ or does not intersect with any interval from $U$. Consequently, the set $U' \cap U$ contains only some complete nonempty intervals from $U'$. The length of a longest interval in $U' \cap U$ is reduced by a factor of 2 or more compared to the length of a longest interval in $U$. Since removing an interval from a collection does not invalidate the properties, the collections $V, U_1^0, \dots, U_1^{k_1}, \dots\dots, U_p^0, \dots, U_p^{k_p}$ are regular and the collections $U_i^0, \dots, U_i^{k_i}, V$ are monotonic.

The second case is when $k < k'$. We can carry out an analysis similar to the one presented in the previous paragraph. The set $U \cap U'$ contains only some complete nonempty intervals from $U$, and regularity and monotonicity hold. However, we do not guarantee that the length of a longest interval in $U$ is reduced, because it could happen that any interval from $U$ is included in an interval from $U'$.

The final case is when $k = k'$. Let us again investigate the result of the operation $U \cap U'$. Take any interval $I$ from $U$, and let us see what part of this interval is contained in $U \cap U'$, if any. By property (ii), for any interval $J$ from $U'$ we have either $I \subseteq J$, or $I \supset J$, or $I \cap J = \emptyset$. If the first subcase occurs, then we are guaranteed that complete $I$ is contained in $U \cap U'$. Suppose that the first subcase does not happen, and so for all $J$, either $I \supset J$ or $I \cap J = \emptyset$ (but never $I \subseteq J$). The result now depends on how many distinct $J$ there are that satisfy $I \supset J$. Suppose that $I \supset J$. Our assumption about the length of intervals ensures that the length of such $J$ is $2^{k-1}$ and of $I$ is $2^k$. By property (i), we can have either zero, or one, or two intervals in $U'$ that are strict subsets of $I$. In the former situation all intervals $J$ have empty intersection with $I$, and so $I$ is not contained in $U \cap U'$. In the latter situation the two intervals combined must yield $I$, and so complete $I$ is contained in $U \cap U'$. The discussion presented in this paragraph so far implies that regularity and monotonicity trivially hold because the intervals contained in $U \cap U'$ are complete intervals from $U$. In the middle scenario, by property (iii), the interval $I$ is partitioned into two halves: $J$ and $I \setminus J$, and so only the half $J$ is contained in $U \cap U'$, but not the other half. An argument similar to that in Lemma 3.1 shows that monotonicity is preserved and another similar to that in Lemma 3.2 shows that regularity is preserved. ∎

The operation of intersection defined in the preceding lemma yields a certain reduction of size or length of the intervals in the resulting collection $V$, compared to the given collection $U$.

COROLLARY 3.4. *If $V \neq U$, then either*

(i) *the length of a longest interval in $V$ is at most half of the length of a longest interval in $U$, i.e.,* $\max_{I \in V} |I| \leq 1/2 \cdot \max_{I \in U} |I|$, *or*

(i) *the total number of elements in the intervals of $V$ is at most the total number of elements in the intervals of $U$ minus half of the length of a longest interval in $U$, i.e.,* $|\cup_{I \in V} I| \leq |\cup_{I \in U} I| - 1/2 \cdot \max_{I \in U} |I|$.

*Proof.* The analysis again proceeds by considering the relationship between the lengths of longest intervals in $U$ and $U'$. We consider three mutually exclusive cases. Suppose that $k > k'$. Then, by Lemma 3.3, the length of a longest interval in $V$ is at most half of the length of a longest interval in $U$. If $k < k'$, then $V$ contains some complete intervals from $U$, and since $U \neq V$, one of them must be missing in

$V$. This missing interval has length at least half of the length of a longest interval in $U$. If $k = k'$, then $V$ contains complete intervals from $U$ or their halves. Again, since $U \neq V$, either one interval or its half is missing. If a half is missing, then this half must contain at least $2^{k-1}$ elements. ∎

From now until the end of section 3.2, let us fix the values of the collections $U_1^0, \ldots, U_1^{k_1}, \ldots\ldots, U_p^0, \ldots, U_p^{k_p}$ as well as the values of variables $i$, $j$, and $m$. Let $V$ be the collection defined in Lemma 3.3 for the fixed values of the collections and the variables. We let $I$ be a fixed interval from the collection $U_i^{k_i}$, and $J$ a fixed interval from the collection $U_j^m$, such that these two intervals have nonempty intersection. This intersection contains a cell $x$ where processor $i$ collides with a different processor $j$. The lemmas and corollaries in the remainder of section 3.2 show transformations on collections that preserve regularity and monotonicity, and lead to certain reductions in the number of elements contained in the intervals of the resulting collections. The proofs are similar to the proofs of Lemma 3.3 and Corollary 3.4 (see Figure 6 for illustrations).

LEMMA 3.5. *Let $D$ be a possibly empty prefix of $I$ and $D'$ a possibly empty suffix of $J$, or conversely, let $D$ be a suffix and $D'$ a prefix, such that $D \cap D' = \emptyset$ and $D \cup D'$ is a nonempty interval. Let $W$ be the collection*

$$W = V \setminus (D \cup D') := \{\ H \mid H \neq \emptyset \ \wedge\ K \in V \ \wedge\ H = K \setminus (D \cup D')\ \}.$$

*Then the number of intervals in $W$ is at most the number of intervals in $V$. If $k \neq k'$, then the collection $W$ contains some complete intervals from $V$. If $k = k'$, then $W$ contains some complete intervals from $V$ and a single half for each of some other intervals from $V$. The collections $W, U_1^0, \ldots, U_1^{k_1}, \ldots\ldots, U_p^0, \ldots, U_p^{k_p}$ are regular, and the collections $U_i^0, \ldots, U_i^{k_i}, V, W$ are monotonic.*

*Proof.* To prove the lemma, we take any interval $K$ from $V$ and argue about what the result of subtracting $D \cup D'$ from $K$ is. We arrange the argument in three cases by the relationship between the length of a longest interval in $U$ and in $U'$.

For the first case suppose that $k > k'$. Then, by Lemma 3.3, the collection $V$ contains only some complete intervals from $U'$. Inspecting the possible lengths of intervals from $U$ and $U'$ reveals that it cannot happen that an interval from $U$ is a strict subset of some interval from $U'$, and so, by property (ii), $J \subseteq I$ (recall that we assume that $I \cap J \neq \emptyset$). Similarly, for any $K$ from $V$, $K \subseteq I$ or $K \cap I = \emptyset$. Note that the interval $D \cup D'$ starts at a tip of $I$, runs through entire $J$, and ends at the opposite tip of $J$. Thus the three sets $(D \cup D') \setminus J$, $J$, and $I \setminus (D \cup D')$ are disjoint possibly empty intervals whose union is $I$. If $K$ does not intersect with $I$, then $K \setminus (D \cup D') = K$. Assume now that $K$ is a subset of $I$. Since $J$ and $K$ are intervals from $U'$, we have, by property (i), that either $J = K$ or $J \cap K = \emptyset$. In the first subcase $K \setminus (D \cup D') = \emptyset$, while in the second subcase $K$ either belongs to $(D \cup D') \setminus J$ or to $I \setminus (D \cup D')$ and so $K \setminus (D \cup D')$ is equal to $\emptyset$ or $K$, respectively. Thus $W$ is equal to $V$ except for possibly some complete intervals removed, and so desired regularity and monotonicity hold.

A symmetric case is when $k < k'$. Now collection $V$ contains only some complete intervals from $U$, and it must be that $I \subseteq J$, and that for any $K$ from $V$, $K \subseteq J$ or $K \cap J = \emptyset$. As above, if $K$ does not intersect with $J$, then $K \setminus (D \cup D')$ is equal to $K$, while if $K \subseteq J$, then $K \setminus (D \cup D')$ is either empty or equal to $K$, and so desired regularity and monotonicity hold.

Finally, consider the case when $k = k'$. Since $I \cap J \neq \emptyset$, by property (ii), we have three subcases: $J \subset I$, $I \subset J$, $I = J$. We consider them in turn. For the first

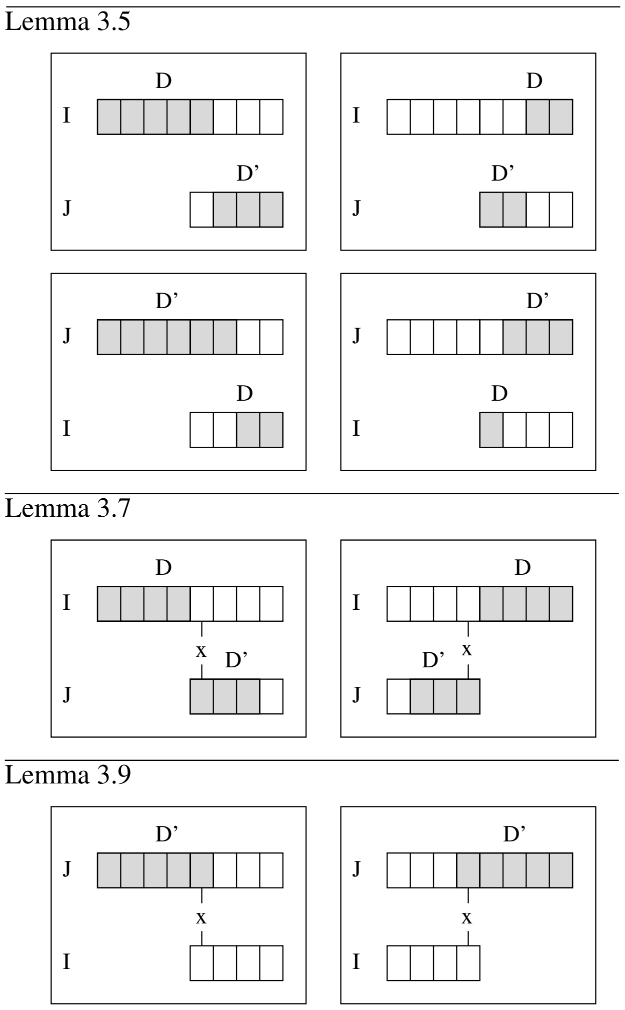


FIG. 6. *This figure illustrates the assumptions of Lemmas* 3.5, 3.7, *and* 3.9.

subcase suppose that $J \subset I$. Since, by property (i), the length of $I$ and $J$ can be either $2^k$ or $2^{k-1}$, the length of $I$ is $2^k$ and, by property (iii), $J$ is a half of $I$ and has length $2^{k-1}$. Thus $D \cup D'$ is either equal to $I$ or equal to a half of $I$. Take any $K$ from $V$. By Lemma 3.3, $V$ contains complete intervals from $U$ or halves of some other intervals form $U$ but never two halves of the same interval from $U$. If $K$ is an interval from $U$, then, by property (i), $K$ is either equal to $I$ or has empty intersection with $I$. If $K \cap I = \emptyset$, then $K \setminus (D \cup D') = K$. If $K = I$, then $K \setminus (D \cup D')$ is equal to either a half of $K$ or an empty set depending on whether $D \cup D'$ is the other half of $K$ or not. If $K$ is a half of an interval from $U$, then either $K \cap I = \emptyset$ or $K$ is a half of $I$. If $K$ is a half of $I$, then $K \setminus (D \cup D')$ is either $K$ or an empty set. Thus in the first subcase the set $K \setminus (D \cup D')$ has length either $2^k$ or $2^{k-1}$ or 0 and is equal to $K$, or a half of $K$, or the empty set. Hence desired regularity and monotonicity hold. For the second subcase suppose now that $I \subset J$. Then $J$ has length $2^k$, $I$ is its half of length $2^{k-1}$, and $D \cup D'$ is equal to $J$ or $I$. Take any $K$ from $V$. Since $K$ is an interval from $U$ or its half and has length $2^k$ or $2^{k-1}$, $K$ is too short to be a strict superset of $J$, and, by property (ii), $K \subseteq J$ or $K \cap J = \emptyset$. When the latter is true, $K \setminus (D \cup D') = K$. Let $K \subseteq J$. If the length of $K$ is $2^k$, then $K = J$ and $K \setminus (D \cup D')$ is equal to $\emptyset$ or a half of $K$. If the length of $K$ is $2^{k-1}$, then $K \setminus (D \cup D')$ is equal to $K$ or $\emptyset$. Thus in the second subcase the set $K \setminus (D \cup D')$ has length either $2^k$ or $2^{k-1}$ or 0. Hence desired regularity and monotonicity hold. Finally, consider the last subcase when $I = J$. Then the set $K \setminus (D \cup D')$ is either empty or equal to $K$, and so it has length either $2^k$, or $2^{k-1}$, or 0, and so desired regularity and monotonicity hold. □

COROLLARY 3.6. *If $U = V$, then the total number of elements in the intervals of $W$ is at most the total number of elements in the intervals of $U$ minus half of the length of a longest interval in $U$, i.e., $|\cup_{I \in W} I| \leq |\cup_{I \in U} I| - 1/2 \cdot \max_{I \in U} |I|$.*

*Proof.* If $U = V$, then the $I$ used in the statement of Lemma 3.5 belongs to $V$. Consequently, one of the sets $K$ from the statement of Lemma 3.5 is equal to $I$. We now follow the last three paragraphs of the proof of Lemma 3.5 to see what the difference is between $V$ and $W$. It cannot be that $k > k'$, because then $U \neq V$. If $k < k'$, then the set $D \cup D'$ used in the statement of Lemma 3.5 contains $I$, and so the collection $W$ does not contain the interval $I$, which has length at least half of the length of the longest interval is $U$. If $k = k'$, then we have the following: when $J \subset I$, then $I$ has the length of a longest interval in $U$ and the set $D \cup D'$ is at least a half of $I$, which is removed; when $I \subset J$, then $D \cup D'$ contains $I$, and so $I$ is removed; when $I = J$, then $I$ is removed. □

LEMMA 3.7. *Let $D \neq \emptyset$ be a prefix of $I$ and let $D' \neq \emptyset$ be a prefix of $J$ such that $x$ is the smallest element in $D'$ and $x - 1$ is the largest element in $D$, or conversely, let $D \neq \emptyset$ be a suffix of $I$ and let $D' \neq \emptyset$ be a suffix of $J$ such that $x$ is the largest element in $D'$ and $x + 1$ is the smallest element in $D$. Let $Q$ be the collection*

$$Q = V \setminus D := \{ \ H \ | \ H \neq \emptyset \ \wedge \ K \in V \ \wedge \ H = K \setminus D \ \}.$$

*Then the number of intervals in $Q$ is at most the number of intervals in $V$. If $k \neq k'$, then $Q$ contains some complete intervals from $V$. If $k = k'$, then $Q$ contains some complete intervals from $V$ and a single half for each of some other intervals from $V$. The collections $Q, U_1^0, \ldots, U_1^{k_1}, \ldots, U_p^0, \ldots, U_p^{k_p}$ are regular, and the collections $U_i^0, \ldots, U_i^{k_i}, V, Q$ are monotonic.*

*Proof.* We begin by showing that there are just two cases to consider. Since $I \cap J \neq \emptyset$, by property (ii), we know that either $I \subseteq J$ or $J \subset I$. Suppose that $I \subseteq J$. Then, when $D'$ is a prefix of $J$, $x$ must be the smallest element in $J$, and so $x - 1$ does not belong to $J$ and so cannot belong to $I$ either, while we know that $x$ belongs to $I$, or when $D'$ is a suffix of $J$, $x$ is the largest element in $J$ and so $x + 1$ does not belong to $J$ and so cannot belong to $I$ either. Hence the assumption that $I \subseteq J$ leads to a contradiction. Consequently it must be that $J \subset I$, and so either $k = k'$ or $k > k'$.

For the first case suppose that $k > k'$. Then $V$ contains only some complete intervals from $U'$. Take any $K$ from $V$. By property (ii), we have one of the three subcases: $K \subseteq I$, $K \supset I$, $K \cap I = \emptyset$. The interval $K$ is too short to be a strict superset of $I$, so the middle subcase cannot happen. If the last subcase happens, then $K \setminus D = K$. Let us now focus on the first subcase when $K \subseteq I$. Notice that the sets $D$, $J$, and $I \setminus (D \cup J)$ are disjoint intervals and their union is $I$. By property (i), either $K = J$ or $K \cap J = \emptyset$. If $K = J$, then $K \setminus D = K$, while when $K$ does not intersect with $J$, then $K \setminus D$ is either empty or equal to $K$. Consequently the collection $Q$ contains only some complete intervals from $V$, and so desired regularity and monotonicity hold.

Finally, consider the second case when $k = k'$. Take any $K$ from $V$. The collection $V$ contains some complete intervals from $U$ or halves of some other intervals from $U$, but never two halves of any interval from $U$. Hence, by property (i), either $K \cap I = \emptyset$, or $K = I$, or $K$ is a half of $I$, but then there is no other interval in $V$ that is equal to the other half of $I$. In the first subcase $K \setminus D = K$, and so let us focus on the two remaining subcases. Note that the length of $J$ can be either $2^k$ or $2^{k-1}$, but since $J \subset I$, the length of $J$ must be $2^{k-1}$, and $J$ must be a half of $I$. As a result, $D$ and $J$ are the two halves of $I$. So if $K = I$, then $K \setminus D$ is equal to $J$, a half of $K$. If $K$ is a half of $I$, then $K \setminus D$ is either $K$ or empty. Again regularity and monotonicity hold for $Q$. $\square$

COROLLARY 3.8. *If $U = V$, then the total number of elements in the intervals of $Q$ is at most the total number of elements in the intervals of $U$ minus half of the length of a longest interval in $U$, i.e., $|\cup_{I\in Q} I| \le |\cup_{I\in U} I| - 1/2 \cdot \max_{I\in U} |I|$.*

*Proof.* As explained in Corollary 3.6, we have that $k \le k'$ and $I$ is in $V$, and so $I = K$ for some $K$ from the statement of the Lemma 3.7. It cannot be that $k < k'$ because this is disallowed by the proof of Lemma 3.7. Thus the only possible relationship between $k$ and $k'$ is that $k = k'$. In this case $I$ has length $2^k$ and a half of $I$ is removed. $\square$

LEMMA 3.9. *Let $x$ be the smallest element in $I$ and $D' \ne \emptyset$ be a prefix of $J$ such that $x$ is the largest element in $D'$, or conversely, let $x$ be the largest element in $I$ and $D' \ne \emptyset$ be a suffix of $J$ such that $x$ is the smallest element in $D'$. Let $R$ be the collection*

$$R = V \setminus (D' \setminus \{x\}) := \{\ H \mid H \ne \emptyset \ \wedge\ K \in V \ \wedge\ H = K \setminus (D' \setminus \{x\})\ \}.$$

*Then the number of intervals in $R$ is at most the number of intervals in $V$. The collection $R$ contains some complete intervals from $V$. The collections $R, U_1^0, \ldots, U_1^{k_1}, \ldots, U_p^0, \ldots, U_p^{k_p}$ are regular, and the collections $U_i^0, \ldots, U_i^{k_i}, V, R$ are monotonic.*

*Proof.* The argument is similar to that of Lemma 3.5: we take an interval $K$ from $V$ and argue what part of the interval is in $R$. We start with an observation that when $D'$ contains just one element $x$ then the result is trivial because $R = V$. Assume therefore that $D'$ contains at least two elements. But then $J$ contains an element that is not in $I$ and so, by property (ii), $I \subset J$. Consequently, we have just two cases to consider: $k = k'$ and $k < k'$. We will study them in turn.

For the first case suppose that $k = k'$. Take any $K$ from $V$. The collection $V$ contains some complete intervals from $U$ or halves of some other intervals from $U$. Hence, by property (i), either $K \cap I = \emptyset$, or $K = I$, or $K$ is a half of $I$. Note that the length of $K$ can be either $2^k$ or $2^{k-1}$ and the length of $I$ is $2^{k-1}$, so $K$ cannot be a half of $I$. As a result, the last subcase does not happen and we have either $K \cap I = \emptyset$ or $K = I$. If $K = I$, then $K \setminus (D' \setminus \{x\}) = K$, while if $K \cap I = \emptyset$, then $K \setminus (D' \setminus \{x\})$ can be either $K$ or empty. Again, regularity and monotonicity hold for $R$.

Finally, consider the second case when $k < k'$. This case is very similar to the case when $k > k'$ in the proof of Lemma 3.7. We shall show it here for completeness. Then $V$ contains only some complete intervals from $U$. Take any $K$ from $V$. By property (ii), we have one of the three mutually exclusive subcases: $K \subseteq J$, $K \supset J$, or $K \cap J = \emptyset$. The interval $K$ is too short to be a strict superset of $J$ so the middle subcase cannot happen. If the last subcase happens, then $K \setminus (D' \setminus \{x\}) = K$. Let us now focus on the first subcase when $K \subseteq J$. Notice that the sets $D' \setminus \{x\}$, $I$, and

$J \setminus ((D' \setminus \{x\}) \cup I)$ are disjoint intervals and their union is $J$. By property (i), either $K = I$ or $K \cap I = \emptyset$. If $K = I$, then $K \setminus (D' \setminus \{x\}) = K$, while when $K$ does not intersect with $I$, then $K \setminus (D' \setminus \{x\})$ is either empty or equal to $K$. Consequently, the collection $R$ contains only some complete intervals from $V$, and so desired regularity and monotonicity hold. ∎

COROLLARY 3.10. *If $U = V$ and $R \neq V$, then the total number of elements in the intervals of $R$ is at most the total number of elements in the intervals of $U$ minus half of the length of a longest interval in $U$, i.e., $|\cup_{I\in R} I| \leq |\cup_{I\in U} I| - 1/2 \cdot \max_{I\in U} |I|$.*

*Proof.* Since $U = V$, then each interval $K$ from the statement of the Lemma 3.9 has length at least half of the length of a longest interval in $U$. But $R \neq V$ and so at least one interval or its part must have been removed. Lemma 3.9 shows that a complete interval is either removed or not a part of it at all. Thus at least one $K$ is missing in $R$ compared to $V$. ∎

## 3.3. Analysis of the algorithm.

This section presents an analysis of the collision algorithm given in Figure 2. Line numbers refer to the code in the figure. Recall that $n$ and $p$ are powers of 2. Without loss of generality, we assume that RMW can transfer $O(p)$ cells between local and shared memory (this assumption can be easily relaxed to comply with our model by first transferring the $O(p)$ cells to a dedicated region of shared memory and then making RMW store a pointer to this region in a cell of the trace table; see the proof of Theorem 3.16 for details).

The basic idea of the proof is to "convert" any asynchronous execution of the algorithm into collections of intervals and then reason about the collections. Any processor iterates through the big while loop (lines `03` to `29`) possibly many times. A brief inspection of the code of the algorithm reveals that at the beginning of each iteration, the collection $U$ that the processor maintains in its local memory contains intervals of cells of the Write-All array and that one of the intervals has an unmarked tip. Intuitively, the cells contained in the intervals are the only ones that may still need to be written to because all other cells have already been written to. An external observer can record (or "remember") the value of the collection $U$ of each processor as the processors iterate, and then reason about the properties of all recorded collections, to conclude that a certain bound on work must hold.

Formally, each time a processor $i$ executes line `03`, we record the value of the local variable $U$ of the processor (the collection $U$ does not change in line `03`, but we can still record $U$). This gives rise to a sequence $U_i^0, U_i^1, U_i^2, \ldots$ of collections of intervals, where $U_i^k$ is the value of collection $U$ of processor $i$ recorded in line `03` of the iteration number $k+1$ of the while loop, or $U_i^k = \emptyset$ when processor $i$ does not reach iteration $k+1$, in the given execution. A convenient way to think about the superscript $k$ is that $U_i^k$ is the collection $U$ of processor $i$ recorded immediately *after* iteration $k$ has been completed by the processor—this is why we start the sequence of superscripts from zero. For example, $U_i^0$ is the value of the collection $U$ recorded at the beginning of the first iteration (immediately after "iteration zero" has been completed). By inspecting the code, we see that $U_i^0$ is always equal to $\{[0, n-1]\}$ for any $i$, because when processor $i$ executes line `03` for the first time, the only instructions that it has executed earlier are these in lines `01` and `02`. Processor $i$ may or may not halt in a given execution. If it does not halt, then the value $U_i^k$ is well defined by the first segment of the definition (processor $i$ will execute line `03` for iteration number $k+1$, for all $k \geq 0$). However, if the processor halts in iteration number $k+1 < \infty$ for some $k \geq 0$, then $U_i^k$ is the last collection recorded for processor $i$, and line `03` will not be reached in iterations $k+2, k+3, \ldots$. Then the second segment of the definition

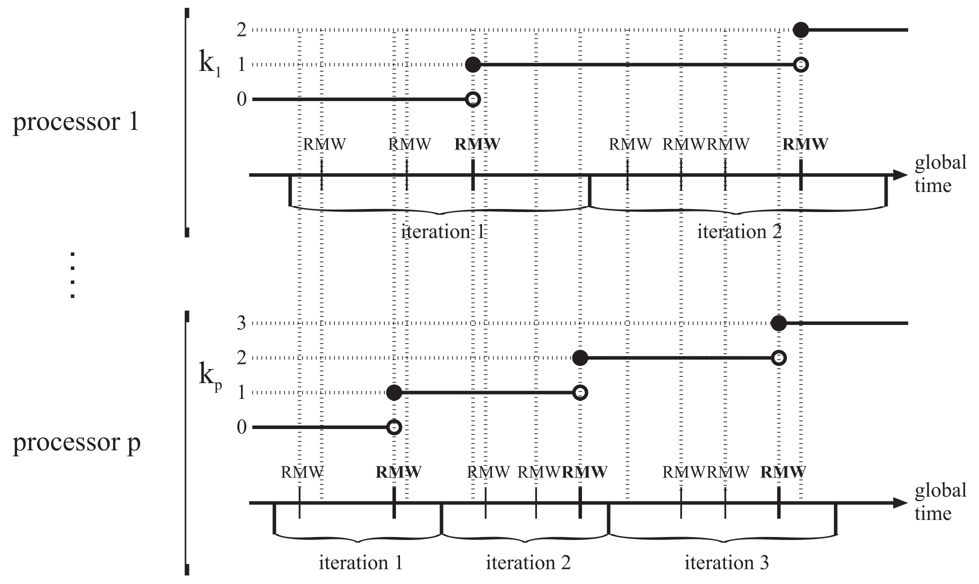


FIG. 7. *The value $k_i$ for processor $i$ increases at the moment when the last RMW instruction is executed in any iteration of the while loop that this processor executes. The induction of Lemma* 3.11 *focuses on these moments.*

puts $U_i^r = \emptyset$ for all $r \geq k+1$. So the sequences of collections are well defined for any execution and any processor.

We introduce additional terminology and notation used in the analysis of the algorithm. We say that a processor *is working on an interval* when it is executing its first RMW in the for loop (lines `06` to `10`) or any instruction during this loop until the processor has executed the last RMW in the for loop. At the moment when this last RMW has been executed, we say that the processor *has completed working on an interval.* Note that this last RWM can be executed either successfully or not. Also, during each iteration of the while loop, the processor may be working on a different interval than in other iterations. For a fixed execution, processor, and iteration of the while loop, we let $U^z$ denote the value of $U$ right before the processor executes line number $z$ of the iteration and $U_z$ denote the value of $U$ immediately after line number $z$. It will be clear from the context which execution, processor, and iteration $U^z$ or $U_z$ refer to.

The analysis starts with three lemmas and a corollary that reduce the analysis of the algorithm to the analysis of properties of collections of intervals. The first lemma shows that the collections of intervals, recorded over time as the algorithms unfolds, have specific structure.

LEMMA 3.11. *Consider any moment (of the global clock) during an execution of the algorithm, and let $k_i \geq 0$ for $1 \leq i \leq p$ be the number of times processor $i$ has completed working on an interval at that time. Then each processor $i$ either will begin work on an interval from $U_i^{k_i}$, or is working on an interval from $U_i^{k_i}$, or will halt without doing any more RMW, or has halted. The intervals $U_1^0, \ldots, U_1^{k_1}, \ldots, U_p^0, \ldots, U_p^{k_p}$ are regular, and the intervals $U_i^0, \ldots, U_i^{k_i}$ are monotonic for any $1 \leq i \leq p$.*

*Proof.* The proof is by induction on the moments in the execution when the values of $k_i$'s change (see Figure 7). We shall see that the lemma holds from the moment when processors begin execution until, but excluding, the first time any processor has completed working on an interval. Then we will consider two moments: a moment immediately before any moment when a processor $i$ has completed working on an interval, and the moment when the processor has completed working on the interval. Thus the value of $k_i$ increases by one from the first moment to the second, and no processor executes any action between these moments. We will argue that if the hypothesis holds at the first moment, it holds at the second moment as well, and also later, until immediately before the next time when a processor (possibly different than $i$) has completed working on an interval.

Let us consider the base case. The lemma is true immediately before, for the first time, a processor has completed working on an interval. Indeed, the first line of the collision algorithm for any processor $i$ sets $U$ to $\{[0, n-1]\}$, and the value of $U$ is not changed at least until the processor reaches line `11` of the first iteration of the while loop. Thus $U_i^0 = \{[0, n-1]\}$ for any $1 \leq i \leq p$. At the moment immediately before a last RMW is executed for the first time by any processor, each processor $i$ either is working or will work on an interval from $U_i^0$. Note that collections $U_1^0, \dots, U_p^0$ are regular, as each contains the same single interval of length that is a power of two. Trivially, the collection $U_i^0$ is monotonic for any $1 \leq i \leq p$ because any single collection is always monotonic.

For the inductive step, pick a moment immediately before any moment when a processor has completed working on an interval, and assume that the lemma is true then. Let this be processor $i$, and let $k_h \geq 0$ denote the number of times processor $h$ has competed working on an interval. Processor $i$ is executing iteration number $k_i + 1$ of the while loop and is about to complete working on an interval for the $(k_i + 1)$-th time. When processor $i$ has finally completed working on the interval, we have two cases: either the processor has succeeded on the last RMW or has failed.

*Case* 1. If processor $i$ succeeds on the last RMW, then it means that the processor has successfully RMWed to all cells in the interval $I$ that it has been working on. Notice that the processor never reads any memory cell that could be written to by a different processor between now and the moment when the processor reaches line `03` again in the next iteration number $k_i + 2$ of the while loop, if the processor ever reaches this line again. Therefore, the value of $U_i^{k_i+1}$ is already determined. Since the processor has been working on an interval $I$ from $U = U_i^{k_i}$, then, in line `12`, the processor removes the interval from the collection. Let $V$ denote the resulting collection of intervals. Clearly, the collections $U_i^0, \dots, U_i^{k_i}, V$ are monotonic and the collections $V, U_1^0, \dots, U_1^{k_1}, \dots, U_p^0, \dots, U_p^{k_p}$ are regular.

When processor $i$ reaches line `24`, the collections $U^{24}, U_1^0, \dots, U_1^{k_1}, \dots, U_p^0, \dots, U_p^{k_p}$ are regular, and the collections $U_i^0, \dots, U_i^{k_i}, U^{24}$ are monotonic. By property (i), if $U^{24}$ contains an interval of length 1, then $U^{24}$ contains intervals of length 1 or 2 only. Hence a possible removal of intervals of length 1 preserves regularity and monotonicity. If $U^{25}$ is empty, then processor $i$ halts and $U_i^{k_i+1} = \emptyset$. Suppose that $U^{25}$ is not empty. The processor will begin work on an interval from a collection, as explained next. When not all tips of intervals in $U^{25}$ are marked, then $U_i^{k_i+1} = U^{25}$ and the processor will begin work on an interval with an unmarked tip from the collection. Alternatively, all tips of intervals in $U^{25}$ are marked. Then $U^{25}$ cannot contain intervals of length 1, because these were just removed. Also $U^{25}$ is not empty. So it contains at least one interval that has length at least 2. Let $R$ be a collection equal to $U^{25}$ except for a longest interval replaced by its two halves (lines `27` to `29`). By Lemma 3.2, the collections $R, U_1^0, \dots, U_1^{k_1}, \dots, U_p^0, \dots, U_p^{k_p}$ are regular, and, by Lemma 3.1, the collections $U_i^0, \dots, U_i^{k_i}, R$ are monotonic. Then $U_i^{k_i+1} = R$. Note that at any moment until the next time a processor will have completed working on an interval, properties related to processors other than $i$ carry over from the inductive assumption. That is, any processor $j \neq i$ will begin work on an interval from $U_j^{k_j}$, or is working on an interval from $U_j^{k_j}$, or will halt without doing any more RMW, or has halted, while processor $i$ will begin work on an interval from $U_i^{k_i+1}$, or is working on an interval from $U_i^{k_i+1}$, or will halt without doing any more RMW, or has halted.

The argument presented so far ensures that the inductive step holds in the case when processor $i$ succeeds on the last RMW.

*Case* 2. Suppose that processor $i$ fails on RMW to a cell $x$ that belongs to the interval $I$ that the processor has been working on. Then the processor reaches line `14` and, if the processor was working on $I$ from left to right, the set $D$ contains a prefix of $I$ up to, but excluding, $x$, or, if the processor was working from right to left, a suffix of $I$ up to, but excluding, $x$. Suppose that it was a processor $j$ that has done a successful RMW to the cell $x$. This must have happened when processor $j$ was working on an interval $J$ either from its left tip toward its right tip or vice versa, and so the set $D'$ read by processor $i$ in line `14` of iteration $k_i + 1$ is a prefix of $J$ ending at $x$, or conversely it is a suffix of $J$ ending at $x$. Note that $i \neq j$ because if a processor does a successful RMW to a cell, then it never again even attempts to do RMW to this cell. Thus the collection $U'^{15}$ is equal to $U_j^m$ for some $0 \leq m \leq k_j$.

We now consider the changes to $U$ that can happen from line `15` to line `23` and see how these changes affect regularity and monotonicity.

After processor $i$ has executed lines `15` to `17` the desired properties hold. Indeed, by Lemma 3.3, the collections $U^{18}, U_1^0, \dots, U_1^{k_1}, \dots, U_p^0, \dots, U_p^{k_p}$ are regular, and the collections $U_i^0, \dots, U_i^{k_i}, U^{18}$ are monotonic.

The execution of processor $i$ now depends on whether $i$ collided with $j$ when working in opposite directions or the same direction.

First, suppose that processor $j$ was working on its interval $J$ in the opposite direction to the direction in which processor $i$ was working. Then $i$ will execute line `19`. As a result, by Lemma 3.5, the collections $U_{19}, U_1^0, \dots, U_1^{k_1}, \dots, U_p^0, \dots, U_p^{k_p}$ are regular, and the collections $U_i^0, \dots, U_i^{k_i}, U_{19}$ are monotonic, and so the collection $U^{24}$ is equal to $U_{19}$.

Second, assume that processor $j$ was working on $J$ in the same direction as processor $i$ has been on $I$. We now study two subcases depending on the success of $i$ in RMW to any cell in $I$.

Subcase number one is when processor $i$ has managed to successfully RMW to at least one cell in $I$. Then $D \neq \emptyset$ and so the processor will execute line `21`. We now argue that specific relationships must hold between $D$, $D'$, and $x$. If $i$ and $j$ worked to the right, then $x$ cannot be the second or later to the right element of $J$, because then the first element of $J$ would be successfully RMWed by $j$, and so $i$ would have failed on an RMW to an element different than $x$, as ensured by the fact that $i$ has been working on consecutive elements from the interval $I$ and that $D \neq \emptyset$. So $x$ must be the first element of $J$, and so $x$ is the smallest element of $D'$ and $x-1$ is the largest element of $D$. Similarly, if $i$ and $j$ were working to the left, then $x$ is the largest element of $D'$ and $x+1$ is the smallest element of $D$. These relationships ensure that when $i$ has executed line `21`, by Lemma 3.7, collections $U_{21}, U_1^0, \dots, U_1^{k_1}, \dots, U_p^0, \dots, U_p^{k_p}$ are regular, and the collections $U_i^0, \dots, U_i^{k_i}, U_{21}$ are monotonic. Then the collection $U^{24}$ is equal to $U_{21}$.

Subcase number two is when processor $i$ failed on its first RMW to a cell in $I$. Then $D = \emptyset$ and so processor $i$ will execute line 22. Since $i$ has failed on its first RMW, then, when $i$ works to the right, $x$ must be the smallest element in $I$ and $x$ the largest element in $D'$, or, when $i$ works to the left, then $x$ is the largest element in $I$ and $x$ the smallest element in $D'$. As a result, after $i$ has executed line 22, by Lemma 3.9, the collections $U_{22}, U_1^0, \dots, U_1^{k_1}, \dots, U_p^0, \dots, U_p^{k_p}$ are regular, and the collections $U_i^0, \dots, U_i^{k_i}, U_{22}$ are monotonic. Then the collection $U^{24}$ is equal to $U_{22}$.

Let us summarize how the changes to $U$ done by processor $i$ from line 14 to line 23 can affect regularity and monotonicity immediately before $i$ reaches line 24. By the above argument, when processor $i$ fails on the RMW, we know that collections $U^{24}, U_1^0, \dots, U_1^{k_1}, \dots, U_p^0, \dots, U_p^{k_p}$ are regular, and the collections $U_i^0, \dots, U_i^{k_i}, U^{24}$ are monotonic. We can carry out the same analysis as in the case of a successful last RMW described in Case 1 earlier to show that any processor either has halted, will halt, is working, or will begin work on an interval at any moment before the next time some $k_g$ is increased, and that the collections are regular and monotonic as desired.

This completes the inductive step and the proof. ∎

The technique used in the proof of Lemma 3.11 could be called the technique of eventual invariant. We formulate an invariant, take any execution that is a sequence of events, show that there is an event when the invariant holds and that, due to the properties of the asynchronous model, eventually there is another event when the invariant holds again, or the algorithm terminates.

Corollary 3.12. *Consider any moment during an execution of the algorithm and let $k_i \geq 0$ for $1 \leq i \leq p$ be the number of times processor $i$ has completed working on an interval at that time. Then the cells $\{0, \dots, n-1\} \setminus U_i^{k_i}$ of the array $w$ have been set to 1 for any $i$. When a processor sets the flag $f$ to 1 and halts, then each cell of $w$ has been set to 1.*

*Proof.* Using a straightforward inductive argument similar to that given in Lemma 3.11, we can demonstrate that any cell $x \in \{0, \dots, n-1\} \setminus U_i^{k_i}$ has been set to 1. In particular, given any processor and its collection $U$ at any moment of the execution, any cell in $\{0, \dots, n-1\} \setminus U$ is known to have been set to 1, along with tips marked by the processor. For the second part, observe that a processor halts only when its $U$ is empty. ∎

The next lemma shows that any processor has at most $p/2$ intervals in its collection $U$ at any time during execution. The key algorithmic tool that yields this bound is the technique of marking tips of intervals. Recall that a processor splits an interval only when all tips of all intervals in $U$ are marked. So in order to produce many intervals, there must be a moment when the processor has many marked tips. This, however, means that the processor must have collided with many other processors. We can infer that during these collisions, knowledge about progress must necessarily be transferred from one processor to the other processor, and so the processor must necessarily remove a tip, thus preventing the number of intervals from growing too much.

Lemma 3.13. *Consider any moment during an execution of the algorithm and let $k_i \geq 0$ for $1 \leq i \leq p$ be the number of times processor $i$ has completed working on an interval at that time. Then for each processor $i$, the collection $U_i^{k_i}$ has at most $p/2$ intervals. None of the collections can have $p$ marked tips.*

*Proof.* The proof is by induction on the moments in the execution when the values of $k_i$'s change, as in Lemma 3.11.

For the base case, we observe that the lemma is true immediately before, for the first time, a processor has completed working on an interval, because $U_1^0 = \cdots = U_p^0$ are collections, each containing just one interval $[0, n-1]$ with no marked tip.

For the inductive step, pick a moment immediately before any moment when a processor has completed working on an interval. Let this be processor $i$, and let $k_h \geq 0$ be the number of times processor $h$ has completed working on an interval then. Assume that collection $U_h^k$ has at most $p/2$ intervals for any $1 \leq h \leq p$ and $0 \leq k \leq k_h$. Processor $i$ is executing iteration number $k_i + 1$ of the while loop and is about to complete working on an interval for the $(k_i + 1)$-th time.

Let us bound the number of intervals in $U$ when processor $i$ reaches line 24 of this iteration. During the time when $i$ is working on the interval, collection $U$ of $i$ is equal to $U_i^{k_i}$. If the last RMW in this iteration is successful, then when the processor reaches line 24, the number of intervals in $U$ is one fewer than in $U_i^{k_i}$, and so it is at most $p/2 - 1$. If the RMW failed, then the processor must have collided with a different processor $j$, and so the value of the variable $U'_{14}$ of processor $i$ is equal to $U_j^m$ for some $0 \leq m \leq k_j$. By the inductive hypothesis the number of intervals in $U'_{14}$ is at most $p/2$. After the processor $i$ has executed lines 15 to 23, the number of intervals in $U$ can be at most the maximum of the number of intervals in $U_i^{k_i}$ and $U_j^m$, because of Lemmas 3.3, 3.5, 3.7, and 3.9. And so $U^{24}$ has at most $p/2$ intervals.

We now study the evolution of $U$ until processor $i$ reaches line 03 again, if ever. We begin the evaluation with three simple cases. First, if $U_{24}$ is empty, then the result is trivial because $U_i^{k_i+1}$ is empty. Second, if $U_{24}$ has an interval with an unmarked tip, then processor $i$ does not execute lines 27 to 29, and so $U_i^{k_i+1}$ is equal to $U_{24}$. Third, if $U_{24}$ is not empty, all tips of all intervals in $U_{24}$ are marked, and $U_{24}$ has strictly fewer than $p/2$ intervals, then the processor $i$ executes lines 27 to 29. But then splitting an interval increases the number of intervals by exactly one. Consequently, in the third case, $U_i^{k_i+1}$ has one interval more compared to $U_{24}$, and so the number of intervals is bounded by $p/2$. In all three cases the inductive step follows.

The final and most interesting case of the inductive step is when the collection $U_{24}$ has exactly $p/2$ intervals and all their tips are marked. We show that this cannot happen by way of contradiction.

Let the collection $U_{24}$ be composed of the intervals $I_1, \ldots, I_{p/2}$. Since all intervals of length 1 with marked tips were removed in line 24, each of the $p/2$ intervals has length at least 2. Thus the intervals have exactly $p$ distinct tips and the tips are marked.

How can the tips get marked? By inspecting the code we see that a tip $x$ can get marked only after $i$ collides with a processor that worked in the same direction when performing RMW to $x$. Then either $D = \emptyset$ or $D \neq \emptyset$. When $D = \emptyset$, $i$ fails on RMW to tip $s$ in an iteration, and then $s$ remains as a tip and gets marked in this very iteration. When $D \neq \emptyset$, $i$ fails on RMW to $x$ other than $s$ in an iteration, and then $x$ becomes a tip and gets marked in this very iteration. Consequently, in order for any tip $x$ to be marked, processor $i$ must fail on RMW to $x$. So processor $i$ must have failed on RMW to the $p$ distinct tips by the moment it reaches line 24 of iteration $k_i + 1$.

Could any of these tips be successfully RMWed by $i$? Note that if processor $i$ had done a successful RMW to a tip of its interval during the for loop in a $k$th iteration of the while loop, $k \leq k_i + 1$, then this tip (not necessarily entire interval) would have been removed from its collection before the processor $i$ reaches line 24 of the while loop in iteration number $k$, and, by the monotonicity property (iv), no subsequent collection $U$ of processor $i$ can contain the tip. Therefore, when processor $i$ reaches line 24 of the current iteration number $k_i + 1$, all $p$ distinct tips of the intervals $I_1, \ldots, I_{p/2}$ have been successfully RMWed by processors other than $i$.

By the pigeonhole principle, therefore, there is a processor $j$ other than $i$, such that the processor $j$ had successfully RMWed to two of the $p$ tips on which $i$ failed. Let $x$ and $y$ be these two tips, such that $x$ was RMW before $y$ was, according to the total order established by the RMW instructions. Let $j_x \leq j_y$ be the iteration numbers of the while loop of processor $j$ during which the processor did successful RMWs to the two tips $x$ and $y$, respectively. We consider two cases depending on whether $x$ and $y$ were RMW in the same iteration or different iterations of the while loop.

For the first case, suppose that $x$ was RMW in an earlier iteration, i.e., $j_x < j_y$. Then processor $j$ must have removed the cell $x$ from its set $U$ by the end of the iteration $j_x$, and so in all subsequent iterations the collection $U$ of processor $j$ does not contain any interval that contains $x$. So when $j$ performs a successful RMW to cell $y$, it stores a collection there, so that none of the intervals of the collection contains $x$. Subsequently, processor $i$ fails on RMW to cell $y$, and then the processor retrieves the collection stored in the cell. At that time, no interval of the collection $U'_{14}$ of processor $i$ contains element $x$. Hence, after intersection and by monotonicity, the interval $U_{24}$ of the iteration number $k_i + 1$ of processor $i$ does not contain any interval that contains $x$. This is a desired contradiction because we assumed that $x$ is a marked tip of an interval in the collection $U_{24}$.

For the second and final case, assume that $x$ and $y$ were RMW during the same iteration of processor $j$'s while loop, i.e., $j_x = j_y$. Hence when $j$ does RMW to cell $y$, its interval $D$ contains $x$. Subsequently, processor $i$ fails on RMW to cell $y$ and retrieves the interval from the cell. So in the iteration when the retrieval occurs, the $D'_{14}$ of processor $i$ contains two distinct elements $x$ and $y$. Suppose that during this iteration $i$ was working in the same direction as the direction in which $j$ was working when $j$ did a successful RMW to $y$. Assume for a moment that the direction was R (right). Moreover, assume that processor $i$ succeeded in one or more RMW in this iteration. Since $y$ is the only cell on which processor $i$ can fail in this iteration, $i$ must have succeeded on the RMW to cell $y-1$. But this cannot be the case, because $x$ is to the left of $y$ and all cells between $x$ and $y$ inclusive had been successfully RMWed by $j$. A similar contradiction is reached when the direction of work was L (left). Hence, in the iteration when retrieval occurred, it cannot be the case that processor $i$ worked in the same direction as processor $j$, $dir_{14} = dir'_{14}$, and processor $i$ succeeded in an RMW, $D_{14} \neq \emptyset$. As a result, $i$ cannot execute line 21 in this iteration and must execute either line 19 or line 22. Consequently, at least one $x$ or $y$ is removed from $U$. Again, this leads to a contradiction.

The inductive step is completed, which proves the lemma. ∎

During an execution, a processor performs some number of iterations of the while loop. The next lemma shows an upper bound on this number. The lemma is proven by noticing that the sequence $U_i^0, U_i^1, U_i^2, \ldots$ of collections of intervals cannot have too long subsequences of equal collections, because the number of marked tips would increase, and that when two consecutive collections are different, then processor $i$ makes substantial progress on its work. This means that the successive collections quickly become "slimmer and slimmer" and eventually become empty.

LEMMA 3.14. *In any execution of the algorithm any processor performs at most* $p^2 + p\,(2p+1)\log n$ *iterations of the while loop.*

*Proof.* Let us consider any execution, any processor $i$, and the sequence of all collections $U_i^0, U_i^1, U_i^2, \ldots$ that have been recorded for this processor in this execution. Since each collection was recorded, the collection is nonempty. This sequence may perhaps be infinite. We will argue that the sequence of recorded collections cannot have too long subsequences of consecutive equal collections, and that when two subsequent collections are different, then there is substantial difference in the total number of elements that belong to the intervals of the consecutive collections. Hence the sequence of collections "slims fast." As a result, we will be able to conclude that the sequence of recorded collections is finite, and in fact quite short. Thus the number of iterations will be small, too, because the number of iterations is exactly equal to the number of recorded collections.

We begin with an observation that if two consecutive recorded collections are equal, then there is one tip that is unmarked in the previous collection and the tip is marked in the subsequent collection, while all tips already marked in the previous collection remain marked in the subsequent collection. Indeed, suppose that $U_i^k = U_i^{k+1}$. Note that when a processor is working on an interval, then the tip $s$ from which the processor has started the work is unmarked. Recall that $U_i^k$ is recorded at the beginning of iteration $k+1$ and $U_i^{k+1}$ is recorded at the beginning of iteration $k+2$. Since the two collections are equal, and transformations applied to $U$ in the iteration are monotonic, the value of the collection $U$ must stay intact during iteration $k+1$. In particular, $U$ does not change from the moment immediately before line `11` until immediately after line `29`. In order for $U$ to remain the same, the processor must have failed on its first RMW to the tip $s$ of the interval on which $i$ was working, and the failure must have been because a different processor $j$ had successfully RMWed to $s$ while $j$ had been working on an interval in the same direction as $i$ was when $i$ attempted an RMW to $s$ (otherwise the value of $U$ would change). As a result, the tip $s$ that was previously unmarked becomes marked in line `23`. Note that no tip $x$ of any interval in any collection is ever unmarked by a processor, unless a part of the interval that contains $x$ is removed from the collection. Hence the number of marked tips in $U_i^{k+1}$ is one plus the number of marked tips in $U_i^k$.

This leads to an observation that the length of a sequence of consecutive equal collections must be bounded. Suppose that for some $k \geq 0$ and $c \geq 0$, collections $U_i^k, U_i^{k+1}, \ldots, U_i^{k+c}$ have been recorded, and they are equal to $U_i^k = U_i^{k+1} = \cdots = U_i^{k+c}$. By Lemma 3.13, the collection $U_i^k$ has at most $p/2$ intervals and so at most $p$ unmarked tips. Because each subsequent collection in the sequence has one fewer unmarked tip and, by Lemma 3.13, no collection can have $p$ marked tips, $c$ can be at most $p-1$. So in the sequence of recorded collections, there can be at most $p$ consecutive equal collections.

We inspect what must happen when two consecutive collections are different. Take any $k \geq 0$ such that $U_i^k, \ldots, U_i^{k+c}, U_i^{k+c+1}$ have been recorded, and $c$ is the largest number so that $U_i^k = U_i^{k+1} = \cdots = U_i^{k+c}$; thus $U_i^{k+c} \neq U_i^{k+c+1}$. Our plan is to show that the difference between collections $U_i^{k+c}$ and $U_i^{k+c+1}$ is "big." Recall that $U_i^{k+c}$ was recorded at the beginning of iteration $k+c+1$, and $U_i^{k+c+1}$ at the beginning of $k+c+2$. So the value of $U^{11}$ in iteration $k+c+1$ is equal to $U_i^{k+c}$, while the value of $U_{29}$ is equal to $U_i^{k+c+1}$. We consider all the ways in which the execution of the processor $i$ may proceed from line `11` until line `29` of iteration $k+c+1$, and study the changes to $U$.

The plan for the subsequent analysis is to consider the first moment when $U$ changes in lines `11` to `29`. We know that a change must occur somewhere inside these lines. Then, by the properties of the transformations of intervals, we can conclude that a "big" modification must occur, and that this modification must carry over to the end of the iteration, by monotonicity of transformations. Then we can count how many times these modifications can happen in the sequence $U_i^0, U_i^1, U_i^2, \ldots$ until $U_i^k$ becomes empty. In the paragraphs that follow we will be assuming that $U$ does not change throughout larger and larger initial sections of the code of lines `11` to `29`.

If processor $i$ succeeded on the last RMW, then it executes line `12`, where an interval is removed from $U$. By property (i), this interval has length that is equal to at least half of the length of a longest interval in $U_i^{k+c}$. Due to monotonicity of subsequent transformations in the iteration, the total number of elements in the intervals of $U_i^{k+c+1}$ is at most the total number of elements in the intervals of $U_i^{k+c}$ minus half of the length of a longest interval in $U_i^{k+c}$.

Suppose, on the other hand, that the processor failed on the RMW to a cell $x$, and so it executes line `14`. If the execution of lines `15` to `17` leads to a change in the value of $U$, then, by Corollary 3.4, either the length of a longest interval in $U^{18}$ is at most half of the length of a longest interval in $U_i^{k+c}$, or the total number of elements in the intervals of $U^{18}$ is at most the total number of elements in the intervals of $U_i^{k+c}$ minus half of the length of a longest interval in $U_i^{k+c}$. Again, due to monotonicity, this relative difference carries over to the difference between $U_i^{k+c}$ and $U_i^{k+c+1}$.

It is possible that processor $i$ has failed on the RMW to the cell $x$, and the execution of lines `15` to `17` does not change $U$. Then $U^{18} = U_i^{k+c}$. Let $j \neq i$ be the processor that successfully RMWed to the cell $x$ while working on an interval. Now we have two cases: either processor $i$ and $j$ worked in opposite directions or the same direction.

For the first case, assume that the colliding processors $i$ and $j$ worked in different directions. Then, by Corollary 3.6, the total number of elements in the intervals of $U_{19}$ is at most the total number of elements in the intervals of $U_i^{k+c}$ minus half of the length of a longest interval in $U_i^{k+c}$. Again, due to monotonicity, the intervals of $U_i^{k+c+1}$ can only have even fewer elements.

For the second case, assume that $i$ and $j$ worked in the same direction. Then processor $i$ executes either line `21` or `22`. In the former case, the value of $U$ always changes; in the latter it may change or not. Suppose that $U$ changes, i.e., $U^{23} \neq U^{21}$. Then, by Corollaries 3.8 and 3.10, the total number of elements in the intervals of $U^{24}$ is at most the total number of elements in the intervals of $U_i^{k+c}$ minus half of the length of a longest interval in $U_i^{k+c}$. It may, however, happen that $U^{23} = U^{21}$, and then we have that $U^{24} = U_i^{k+c}$. Consequently, if any interval of length 1 is removed in line `24`, then, by property (i), this interval is at least as long as half of the length of a longest interval in $U_i^{k+c}$. Finally, assume that no interval of length 1 is removed. Since $U_i^{k+c+1}$ is recorded, then $i$ does not halt in line `25`, and so $U^{26} = U_i^{k+c}$. Then it cannot be the case that there is an unmarked tip, because we assumed that $U_i^{k+c} \neq U_i^{k+c+1}$. Hence a longest interval in $U^{26}$ gets split, and so $U_i^{k+c+1}$ is equal to $U_i^{k+c}$ except for a longest interval split into two halves.

Let us sum up the above study of the modifications to $U$ that must occur from line `11` to line `29`, when the collection $U_i^{k+c+1}$ is different than the collection $U_i^{k+c}$. There are three possible modifications: either the length of a longest interval in $U_i^{k+c+1}$ is at most half of the length of a longest interval in $U_i^{k+c}$, or $U_i^{k+c+1}$ is equal to $U_i^{k+c}$ except for a longest interval being split into two halves, or the total number of elements in the intervals of $U_i^{k+c+1}$ is at most the total number of elements in the intervals of $U_i^{k+c}$ minus half of the length of a longest interval in $U_i^{k+c}$.

We now count how many times each of these three modifications can happen until

$U$ becomes empty (in which case processor $i$ must halt). Reduction of length by the factor of 2 or more can happen at most $\log n$ times, a longest interval can be split into two halves at most $p \log n$ times (because the number of intervals in a collection is bounded by $p/2$), and at least half of a longest interval can be removed at most $p\,(1 + \log n)$ times. As a result $U_i^{p(\log n + p \log n + p(1+\log n))} = \emptyset$, and the lemma has been proven. ◻

We are now ready to prove the main result of this paper.

THEOREM 3.15. *The algorithm solves the CWA problem and has work of $O(n + p^4 \log n)$; the processors combined set at most $n + 4p^3 \log n$ cells of $w$ to 1.*

*Proof.* Consider any execution of the algorithm. By Lemma 3.13 each processor performs a bounded number of iterations, and, by Corollary 3.12, when a processor stops all cells of $w$ have been set to 1. Hence the algorithm solves the CWA problem.

We now argue about the work complexity of the algorithm. Let us fix a processor and divide each of its iterations of the while loop into two parts. The first part contains the instructions starting from the first RMW of the for loop until but not including the last RMW of the for loop, while the second part contains all other instructions in the iteration (the second part has two discontinuous sections of instructions). Note that all RMW in the first part are successful and so the combined work of all processors on the first parts is $O(n)$ (using a pointer representation of the collections) and at most $n$ cells of the array $w$ are set to 1 by the $p$ processors during the first parts. For any processor, the number of second parts is equal to the number of iterations, which is bounded, by Lemma 3.14, by $p^2 + p\,(2p+1) \log n$. So the total number of iterations performed by all processors is bounded by $p\left(p^2 + p\,(2p+1)\log n\right)$. During each second part at most one cell of $w$ can be set to 1, and so the $p$ processors combined may set to 1 at most $p^3 + p^2\,(2p+1)\log n$ cells of $w$ in their second parts. Recall that, by Lemma 3.13, the number of intervals in any collection during the execution is at most $p/2$ and so any second part takes $O(p)$ instructions to execute. Thus the combined work that processors performed on the second parts is $O(p^4 \log n)$. This completes the proof. ◻

Recall that the code of the algorithm assumes that RMW can transfer $O(p)$ cells between local and shared memory. We now discuss how to relax this assumption to comply with our model which allows only $O(1)$ cells to be transferred. The relaxation can be accomplished using pointer representation of collection $U$.

THEOREM 3.16. *In a single execution of the collision algorithm, the $p$ processors combined use at most $O(n + p^4 \log n)$ cells of shared memory.*

*Proof.* Each processor $i$ maintains a shared memory array where collections $U_i^0, U_i^1, U_i^2, \ldots$ will be placed (see Figure 8). The array has $p/2$ rows and $p^2 + p\,(2p+1)\log n$ columns, and is stored in a dedicated section of shared memory. The location of the section inside shared memory can be selected using the unique identifier of the processor. Each entry of the array stores two numbers representing tips of an interval. Recall that in our model each memory cell can accommodate $O(\log n)$ bits. Hence the total number of shared memory cells sufficient for the $p$ arrays is $O(p^4 \log n)$.

When processor $i$ begins iteration $k$, it transfers the collection $U$ to column $k$ of the array. In the algorithm as stated in Figure 2, each RMW transfers the current collection $U$ to a cell of the trace table *tab*. This violates the model because we may need to transfer $O(p)$ cells while in our model we assume that RMW can access a constant number of cells only. In the actual version of the collision algorithm, the processor will transfer a pointer to column $k$ of the array during any RMW operation

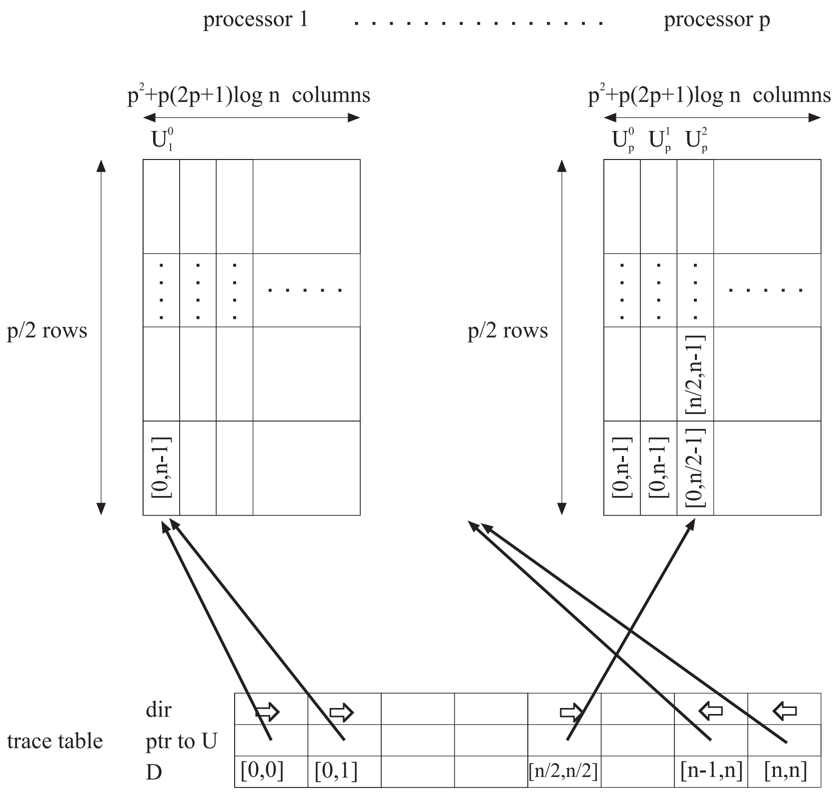


FIG. 8. *Pointer representation of collections of intervals. There are p shared memory arrays, each dedicated to a processor. Collections U generated by a processor are stored in columns of the array. The trace table, also in shared memory, contains pointers to columns.*

of iteration $k$. Thus, assuming that a pointer takes $O(\log n)$ bits, RMW needs only to transfer a constant number of cells. In the algorithm stated in Figure 2, processor reads $U$ directly from the trace table. In the actual algorithm, the processor will read a pointer to a column of an array, and retrieve a collection of intervals from the column.

The number of rows allocated for each processor is sufficient, by Lemma 3.13, to accommodate any $U$ produced during any execution of the algorithm. The number of columns is sufficient, by Lemma 3.14, to complete one execution of the collision algorithm. $\square$

**4. Future work.** It should be possible to tighten the analysis of work complexity of the algorithm by further exploring the information flow between processors. We believe that the actual work complexity of our algorithm is no more than $O(n + p^3 \log n)$ because the analysis given in the proof of Lemma 3.14 appears to have a fair amount of slack.

We believe that it should be possible to further reduce the work and space complexities of the algorithm by modifying data structures. Specifically, instead of creating a new $U$ during each iteration, we could try to reuse the parts of $U$ that have not changed since the prior iteration. This should decrease space complexity, but may increase work complexity as the new representation of $U$ may be more "dispersed." In the proof of Theorem 3.15, we bounded by $O(p)$ the time to perform the transformations of $U$. Is $\Theta(p)$ necessary? It would be interesting to find a representation of $U$ that would allow the performance of $r$ transformations in time $o(rp)$ and space $o(rp)$, possibly by exploiting regularity and monotonicity.

In our algorithm there may be more than one unmarked tip to select from at the beginning of an iteration of the while loop. Currently, the algorithm arbitrarily selects an unmarked tip of an interval. However, one can consider a more refined process of selection when there are choices. One could apply the techniques of "oblivious schedules" of Anderson and Woll [1], in conjunction with identifiers of processors, so as to select a tip to begin work from, so that overall not too many processors also select this tip. The combination of the collision technique and the oblivious technique seems a promising approach for constructing an algorithm that would have asymptotically optimal work for a wider range of the number of processors.

The algorithm uses a special RMW primitive to efficiently detect collisions. Is the use of such a primitive essential for obtaining an algorithm with asymptotically optimal work for a wide range of the number of processors? Would the use of a more standard single word RMW, compare and swap, or even atomic reads and writes be sufficient? A recent paper of Kowalski and Shvartsman [26] shows that atomic reads and writes are sufficient.

Work of any deterministic asynchronous algorithm for the CWA problem with $p \leq n^{1-\epsilon}$ processors must obviously be $\Omega(n)$. We have shown that there is an $O(n)$ algorithm when $1 \geq \epsilon \geq 4/5$. Is it true that there exists an $O(n)$ algorithm for $\epsilon$ arbitrary close to 0? For what values of $p$ compared to $n$ is there a nontrivial lower bound on work of a deterministic algorithm? In particular, is $\omega(n)$ work necessary when $p$ is $o(n/\log n)$?

**Acknowledgments.** The author thanks his host, Charles Leiserson, for an invitation to join the Supercomputing Technologies Group and for discussions, including those on how to write a better proof. This paper was inspired by earlier work with Alex Shvartsman. The author thanks Bogdan Chlebus, Piotr Indyk, Arnold Rosenberg, Alex Russell, Alex Shvartsman, and the anonymous PODC'03 reviewers for reading drafts of the paper and commenting on them. Their input improved the clarity of the presentation. The author particularly acknowledges the three *SIAM Journal on Computing* reviewers for their careful reading of the manuscript, many detailed comments, and suggestions to better explain how the algorithm works. Finally, the author thanks Mumtaz Lohawala for hosting him in her house for one year during the author's visit to MIT.

## REFERENCES

[1] R. J. Anderson and H. Woll, *Algorithms for the certified write-all problem*, SIAM J. Comput., 26 (1997), pp. 1277–1283.

[2] J. Aspnes and M. Herlihy, *Wait-free data structures in the asynchronous PRAM model*, in 2nd Symposium on Parallel Algorithms and Architectures SPAA'90, 1990, pp. 340–349.

[3] Y. Aumann, M. A. Bender, and L. Zhang, *Efficient execution of nondeterministic parallel programs on asynchronous systems*, Inform. and Comput., 139 (1997), pp. 1–16.

[4] Y. Aumann, Z. M. Kedem, K. V. Palem, and M. O. Rabin, *Highly efficient asynchronous execution of large-grained parallel programs*, in 34th IEEE Symposium on Foundations of Computer Science FOCS'93, 1993, pp. 271–280.

[5] Y. Aumann and M. O. Rabin, *Clock construction in fully asynchronous parallel systems and PRAM simulation*, Theoret. Comput. Sci., 128 (1994), pp. 3–30.

[6] A. Baratloo, P. Dasgupta, and Z. M. Kedem, *CALYPSO: A novel software system for fault-tolerant parallel processing on distributed platforms*, in 4th International Symposium on High Performance Distributed Computing HPDC'95, 1995, pp. 122–129.

[7] A. J. Bernstein, *Program analysis for parallel processing*, IEEE Transactions on Electronic Computers, EC-15 (1966), pp. 757–762.

[8] J. Buss, P. C. Kanellakis, P. L. Ragde, and A. A. Shvartsman, *Parallel algorithms with processor failures and delays*, J. Algorithms, 20 (1996), pp. 45–86.

[9] B. Chlebus, S. Dobrev, D. Kowalski, G. Malewicz, A. Shvartsman, and I. Vrto, *Towards practical deterministic Write-All algorithms*, in 13th Symposium on Parallel Algorithms and Architectures SPAA'01, 2001, pp. 271–280.

[10] B. S. Chlebus, A. Gambin, and P. Indyk, *PRAM computations resilient to memory faults*, in 2nd European Symposium on Algorithms ESA'94, 1994, pp. 401–412.

[11] R. Cole and O. Zajicek, *The APRAM: Incorporating asynchrony into the PRAM model*, in 2nd ACM Symposium on Parallel Algorithms and Architectures SPAA'89, 1989, pp. 169–178.

[12] R. Cole and O. Zajicek, *The expected advantage of asynchrony*, in 3rd Annual ACM Symposium on Parallel Algorithms and Architectures SPAA'90, 1990, pp. 85–94.

[13] D. Culler, R. Karp, D. Patterson, A. Sahay, K. E. Schauser, E. Santos, R. Subramonian, and T. van Eicken, *LogP: Towards a realistic model of parallel computation*, in 4th ACM Principles and Practices of Parallel Programming, 1993, pp. 1–12.

[14] P. Dasgupta, Z. M. Kedem, and M. O. Rabin, *Parallel processing on networks of workstations: A fault-tolerant, high performance approach*, in 15th International Conference on Distributed Computing Systems ICDCS'95, 1995, pp. 467–474.

[15] S. Fortune and J. Wyllie, *Parallelism in random access machines*, in 10th Annual ACM Symposium on Theory of Computing, 1978, pp. 114–118.

[16] J. D. Garofalakis, P. G. Spirakis, B. Tampakas, and S. Rajsbaum, *Tentative and definite distributed computations: An optimistic approach to network synchronization*, Theoretical Computer Science, 128 (1–2) (1994), pp. 63–74.

[17] P. B. Gibbons, *A more practical PRAM model*, in 2nd ACM Symposium on Parallel Algorithms and Architectures SPAA'89, 1989, pp. 158–168.

[18] J. F. Groote, W. H. Hesselink, S. Mauw, and R. Vermeulen, *An algorithm for the asynchronous write-all problem based on process collision*, Distributed Computing, 14 (2001), pp. 75–81.

[19] M. Herlihy, *Wait-free synchronization*, ACM Transactions on Programming Languages and Systems, 13 (1991), pp. 124–149.

[20] P. C. Kanellakis and A. A. Shvartsman, *Efficient parallel algorithms can be made robust*, Distributed Computing, 5 (1992), pp. 201–247.

[21] P. C. Kanellakis and A. A. Shvartsman, *Fault-Tolerant Parallel Computation*, Kluwer Academic Publishers, Boston, MA, 1997.

[22] R. M. Karp, M. Luby, and F. Meyer auf der Heide, *Efficient PRAM simulation on a distributed memory machine*, Algorithmica, 16 (1996), pp. 517–542.

[23] Z. M. Kedem, K. V. Palem, M. O. Rabin, and A. Raghunathan, *Efficient program transformations for resilient parallel computation via randomization (preliminary version)*, in 24th Annual ACM Symposium on Theory of Computing STOC'92, 1992, pp. 306–317.

[24] Z. M. Kedem, K. V. Palem, A. Raghunathan, and P. G. Spirakis, *Combining tentative and definite executions for very fast dependable parallel computing (extended abstract)*, in 23rd Annual ACM Symposium on Theory of Computing STOC'91, 1991, pp. 381–390.

[25] Z. M. Kedem, K. V. Palem, and P. G. Spirakis, *Efficient robust parallel computations*, in 22nd Annual ACM Symposium on Theory of Computing STOC'90, 1990, pp. 138–148.

[26] D. R. Kowalski and A. A. Shvartsman, *Writing-All deterministically and optimally using a non-trivial number of asynchronous processors*, in 16th Annual ACM Symposium on Parallelism in Algorithms and Architectures SPAA'04, 2004, pp. 311–320.

[27] D. E. Knuth, *The Art of Computer Programming*, Vol. 3, 3rd ed., Addison-Wesley, Reading, MA, 1998.

[28] C. P. Kruskal, L. Rudolph, and M. Snir, *A complexity theory of efficient parallel algorithms*, Theoret. Comput. Sci., 71 (1990), pp. 95–132.

[29] L. Lovász, *Combinatorial Problems and Exercises*, 2nd ed., North-Holland, Amsterdam, 1993.

[30] G. Malewicz, *A work-optimal deterministic algorithm for the asynchronous Certified Write-All problem*, in 22nd Annual ACM Symposium on Principles of Distributed Computing PODC'03, 2003, pp. 255–264.

[31] G. Malewicz, *A tight analysis and near-optimal instances of the algorithm of Anderson and Woll*, Theoretical Computer Science, 329 (2004), pp. 285–301.

[32] G. Malewicz, *Distributed Scheduling for Disconnected Cooperation*, doctoral dissertation, Computer Science, University of Connecticut, Storrs, CT, 2003.

[33] C. Martel, A. Park, and R. Subramonian, *Work-optimal asynchronous algorithms for*

*shared memory parallel computers*, SIAM J. Comput., 21 (1992), pp. 1070–1099.

[34] C. Martel, and R. Subramonian, *How to Emulate Synchrony*, Technical Report CSE-90-26, UC Davis, Davis, CA, 1990.

[35] C. Martel and R. Subramonian, *Asynchronous PRAM algorithms for list ranking and transitive closure*, in International Conference on Parallel Processing ICPP'90, Vol. 3, 1990, pp. 60–63.

[36] C. Martel and R. Subramonian, *On the complexity of certified write-all algorithms*, J. Algorithms, 16 (1994), pp. 361–387.

[37] J. Naor and R. M. Roth, *Constructions of permutation arrays for certain scheduling cost measures*, Random Structures Algorithms, 6 (1995), pp. 39–50.

[38] R. H. B. Netzer and B. P. Miller, *On the complexity of event ordering for shared-memory parallel program executions*, in the International Conference on Parallel Processing ICPP'90, Vol. 2, 1990, pp. 93–97.

[39] N. Nishimura, *Asynchronous shared memory parallel computation*, SIAM J. Sci. Comput., 23 (1994), pp. 1231–1252.

[40] A. A. Shvartsman, *Achieving optimal CRCW PRAM fault-tolerance*, Inform. Process. Lett., 39 (1991), pp. 59–66.

[41] R. Subramonian, *Designing synchronous algorithms for asynchronous processors*, in 4th Annual ACM Symposium on Parallel Algorithms and Architectures SPAA'92, 1992, pp. 189–198.

[42] L. Valiant, *A bridging model for parallel computation*, Communications of the ACM, 33 (1990), pp. 103–111.

SIAM J. COMPUT. © 2005 Society for Industrial and Applied Mathematics
Vol. 34, No. 5, pp. 1025–1106

# THE BISIMULATION PROBLEM FOR EQUATIONAL GRAPHS OF FINITE OUT-DEGREE*

GÉRAUD SÉNIZERGUES†

**Abstract.** The *bisimulation* problem for equational graphs of finite out-degree is shown to be decidable. We reduce this problem to the $\eta$-bisimulation problem for deterministic rational (vectors of) boolean series on the alphabet of a deterministic pushdown automaton $\mathcal{M}$. We then exhibit a *complete formal system* for deducing equivalent pairs of such vectors.

**Key words.** bisimulation, equational graphs, deterministic pushdown automata, rational languages, finite dimensional vector spaces, matrix semigroups, complete formal systems

**AMS subject classifications.** Primary, 68Q70; Secondary, 03D05, 05C75, 20M35

**DOI.** 10.1137/S0097539700377256

## 1. Introduction.

### 1.1. Motivations.

**Processes.** In the context of concurrency theory, several notions of "behavior of a process" and "behavioral equivalence between processes" have been proposed. Among them, the notion of *bisimulation* equivalence seems to play a prominent role (see [26]). The question of whether this equivalence is *decidable* or not for various classes of infinite processes has been the subject of many works in the last fifteen years (see, for example, [1, 5, 18, 7, 14, 17, 8, 6, 45, 19, 37, 44, 22]).

The aim of this work is to show decidability of the bisimulation equivalence for the class of all processes defined by pushdown automata (pda) whose $\epsilon$-transitions are deterministic and decreasing (of course, we assume that $\epsilon$-transitions are *not* visible, which implies that the graphs of the processes considered here might have infinite in-degree). This problem was raised in [6] (see Problem 6.2 of this reference) and is a significant subcase of the problem raised in [45] (as the bisimulation problem for processes "of type $-1$").

**Infinite graphs.** A wide class of graphs enjoying interesting decidability properties has been defined in [11, 2, 3] (see [12] for a survey). In particular it is known that the problem

$$\textit{are } \Gamma, \Gamma' \textit{ isomorphic?}$$

is decidable for pairs $\Gamma, \Gamma'$ of equational graphs. It seems quite natural to investigate whether the problem

$$\textit{are } \Gamma, \Gamma' \textit{ bisimilar?}$$

is decidable for pairs $\Gamma, \Gamma'$ of equational graphs. We show here that this problem is decidable for equational graphs of finite out-degree.

*Received by the editors August 29, 2000; accepted for publication (in revised form) August 18, 2004; published electronically June 3, 2005. This work has been fully supported by the CNRS, at Bordeaux, in 1996–1998 and by the Humboldt Foundation, at Stuttgart, in 2004.

http://www.siam.org/journals/sicomp/34-5/37725.html

†LaBRI, Université de Bordeaux I, 351, Cours de la Libération 33405 Talence, France (ges@labri.fr).

**Formal languages.** Another classical equivalence relation between processes is the notion of *language* equivalence. The decidability of language equivalence for *deterministic* pushdown automata (dpda) has been established in [34, 40] (see also [36, 35] for shorter expositions of this result). It was first noticed in [1] that, in the case of deterministic processes, language equivalence and bisimulation equivalence are identical. Moreover dpda can always be normalized (with preservation of the language) in such a way that $\epsilon$-transitions are all decreasing. Hence the main result of this work is a generalization of the decidability of the equivalence problem for dpda.

**Mathematical generality.** More precisely, the present work *extends the notions* developed in [34] so as to obtain a more general result. As a by-product of this extension, we obtain a deduction system which, in the deterministic case, seems *simpler* than the one presented in [34] (see system $\mathcal{B}_3$ in section 10).

The present work can also be seen as a common generalization of three different results: the results of [45, 19] establishing decidability of the bisimulation equivalence in two nondeterministic subclasses of the class considered here, and the result of [34] dealing only with deterministic pda (or processes).

**Logics.** Our solution consists in constructing a *complete* formal system, in the general sense taken by this word in mathematical logics; i.e., it consists of a set of well-formed assertions, a subset of basic assertions, the axioms, and a set of deduction rules allowing one to derive new assertions from assertions which are already generated. The well-formed assertions we are considering are pairs $(S, T)$ of rational boolean series over the nonterminal alphabet $V$ of some strict-deterministic grammar $G = \langle X, V, P \rangle$. Such an assertion is true when the two series $S, T$ are bisimilar.

Several simple formal systems generating all the identities between boolean rational expressions have been the subject of several works (see [32, 4, 21]); the case of bisimilar rational expressions has been addressed in [25, 20].

A tableau proof-system generating all the bisimilar pairs of words with respect to a given context-free grammar in Greibach normal form was also given in [18].

Our complete formal systems can be seen as participating in this general research stream ([39] provides an overview of this subject in the context of equivalence problems for pda).

### 1.2. Results.

The main decidability result of this work is the following.

THEOREM 10.7. *The bisimulation problem for rooted equational* 1*-graphs of finite out-degree is decidable.*

The main structural result obtained here is as follows.

THEOREM 10.14. *$\mathcal{B}_3$ is a complete deduction system.*

Here, $\mathcal{B}_3$ is a formal system whose elementary rules just express the basic algebraic properties of bisimulation: the fact that it is an equivalence relation, that it is compatible with right and left (matricial) product, that Arden's lemma remains true modulo bisimulation and, at last, its link with one-step derivation (rule (R34)). Completeness means here that *all* pairs of bisimilar rational "deterministic" boolean series are generated by this formal system.

### 1.3. Overview and roadmap.

**Overview: Large scale.** Let us describe the principal flow of ideas which underpins our proof.

*Step* 1: *Reduction.* We reduce the bisimulation problem for vertices $v, v'$ of an equational graph $\Gamma$ of finite out-degree to an analogous problem, but with more algebraic flavor: given two deterministic rational row-vectors $S, S'$, are they $\eta$-bisimilar?

The precise definition of what is a deterministic rational row-vector is given in section 3.1, the notion of $\eta$-bisimilarity for vectors is defined in section 3.2, and the reduction itself is stated in Lemma 3.25.

*Step* 2: *Logical system.* We define a formal system, named $\mathcal{B}_0$, whose assertions are the pairs of deterministic rational row-vectors $(S, S')$. Such an assertion is considered as true iff the vectors $S, S'$ are $\eta$-bisimilar. The system $\mathcal{B}_0$ is defined in section 4.3 and its soundness is immediately proved there.

*Step* 3: *Strategies for* $\mathcal{B}_0$. We define some suitable notions of *strategies* for a formal system: a strategy is a map sending every partial proof $P$ into a partial proof $P'$ containing $P$. What we have in mind is to build, step by step, a proof for a given true assertion by iteratively applying a strategy. Section 7 is devoted to the construction of such strategies for the system $\mathcal{B}_0$.

*Step* 4: *Completeness of* $\mathcal{B}_0$. We proceed to a close analysis of the "proof-trees" produced by the above strategies in order to show that, starting from a true assertion $(S, S')$, a finite proof-tree is always reached. This succeeds in proving that $\mathcal{B}_0$ is a *sound* and *complete* logical system. Unfortunately, $\mathcal{B}_0$ cannot be easily shown to be *recursively enumerable*, due to one of its rules, namely, rule (R5).

*Step* 5: *Elimination.* It turns out that this problematic rule (R5) can be *eliminated* from $\mathcal{B}_0$, resulting in a smaller system $\mathcal{B}_1$ which is still sound, complete, and recursively enumerable. This logical result allows us to prove decidability of the $\eta$-bisimulation problem for deterministic rational row-vectors, and hence of the bisimulation problem for vertices of an equational graph of finite out-degree (by Step 1).

*Step* 6: *Simplifications.* This last step is useless for decidability purposes but sheds light on the *structure* of bisimulation equivalence. Successive elimination arguments lead us to a fairly simple logical system, named $\mathcal{B}_3$, which is still sound and complete. The rules of $\mathcal{B}_3$ are just consisting of the principal algebraic properties of vectors-$\eta$-bisimulation and a single rule expressing the precise grammar (or process) we are examining (rule (R34)).

**Overview and roadmap: Medium scale.** Let us describe now, section by section, the main successive technical contributions to the general flow described in the previous overview.[1]

*Section* 2.

2.1. We recall the notion of graph-bisimulation as classically stated in the literature.

2.2. As well, we recall the notion of pda.

2.3. We characterize the class of graphs under scrutiny, the "equational 1-graphs of finite out-degree," as the computation-graphs of normalized pda (a precise proof is delayed to the appendix).

2.4. We recall the definition of a deterministic context-free grammar.

2.5. We state the basic definitions allowing us to translate the usual notions of left-derivation with respect to (w.r.t.) (resp., language generated by) a context-free grammar $G$ into a more algebraic framework: everything is formulated now within semirings of boolean series over finite sets of undeterminates,

[1]Sections 1 and 11 do not participate in the proof itself and therefore are left out of this overview. The appendix is also neglected here because of its marginality.

endowed with two right-actions $\odot, \bullet$: the first right-action expresses a one-step left-derivation, while the second right-action is just the well-known residual-action.

*Section* 3.

3.1. We recall the notion of *deterministic boolean series*, which plays a role analogous to that of "configuration" of a dpda. The advantage of this notion is that it allows extensions to vectors and matrices and supports several well-behaved operations.

3.2. We reduce the initial bisimulation problem over graphs to a bisimulation problem over deterministic rational row-vectors (we call this new kind of bisimulation the $\sigma$-$\eta$-bisimulation). For every pair $(S, S')$ of deterministic series, we introduce a notion of word-$\eta$-bisimulation which is a kind of equivalence relation over words that "witnesses" the fact that $(S, S')$ are indeed $\sigma$-$\eta$-bisimilar. Operations on such word-$\eta$-bisimulations are introduced in order to prove, later on, some algebraic properties of $\sigma$-$\eta$-bisimulation.

3.3. The usual notions of derivation and *stacking* derivation are translated in our framework. Some basic properties are demonstrated.

*Section* 4.

4.1. We define a very general notion of *deduction system.* Its only nonstandard aspect, as compared to the usual "Hilbert-style systems," is that the proofs can "loop": it may happen that a correct proof contains an assertion $A$, which is deduced, by means of finitely many steps, from a set of axioms and (though such a feature was not expected) $A$ itself.

4.2. We introduce a general notion of *strategy* w.r.t. a given deduction system: it is just a map sending every partial proof $P$ into a partial proof $P'$ containing $P$. The desirable properties of such strategies are defined there.

4.3. We define here *the* principal deduction system $\mathcal{B}_0$ that we expect to be able to generate exactly the set of $\sigma$-$\eta$-bisimilar pairs of vectors $(S, S')$. We immediately establish that $\mathcal{B}_0$ is sound (Lemma 4.9). Some interesting algebraic corollaries are deduced.

4.4. We isolate a subsystem of $\mathcal{B}_0$, which we name $\mathcal{C}$, that is independent of any graph or automaton: the rules of $\mathcal{C}$ capture the essential algebraic properties relating the $\sigma$-$\eta$-bisimulation relation with the matricial product. We then state four useful general deductions within the system $\mathcal{C}$.

*Section* 5.

5.1. A set of row-vectors which is closed under linear combination is named a *d-space.* Such a d-space, endowed with the right-operation $\odot$, is the key algebraic structure we shall use later. We define a natural notion of *linear independence* for families of vectors and show that it enjoys one of the usual properties of linear independence: if a family is dependent, then one of the vectors of the family is a linear combination of the others.

5.2. Applying repeatedly the above property of dependent families, we exhibit a "triangulation process" for systems of equations: given a system $\mathcal{S}$ of $n$ equations $\sum_{j=1}^{d} \alpha_{i,j} S_j \sim \sum_{j=1}^{d} \beta_{i,j} S_j$ (where $\sim$ denotes the $\sigma$-$\eta$-bisimulation) one can transform $\mathcal{S}$ in such a way that the last equation, which we call the *inverse equation*, combines the coefficients $\alpha_{i,\star}, \beta_{i,\star}$ but does not involve the $S_j$ anymore. The exact relationship between $\mathcal{S}$ and the inverse equation, INV($\mathcal{S}$), is studied there.

In fact, the above description is accurate only when $\eta$ is just the equality relation. In the general case we are lead to introduce the notion of an *oracle*: an oracle is a choice of word-$\eta$-bisimulation for every pair of $\eta$-bisimilar vectors. The "inverse" equation determined by $\mathcal{S}$ and by an oracle $\mathcal{O}$ is denoted by $\mathrm{INV}^{(\mathcal{O})}(\mathcal{S})$.

*Section* 6. Here are collected all the definitions of "constants" used throughout this article: these are all the integers, depending on a given initial automaton $\mathcal{M}$, an equivalence $\eta$, and an initial pair of vectors $(S_0^-, S_0^+)$ which one would like to test for $\sigma$-$\eta$-bisimilarity.

*Section* 7. We define here strategies for the particular deduction system $\mathcal{B}_0$.

7.1. We define several substrategies based on both the algebraic properties established in section 4.3 and the triangulation process studied in section 5.2.

7.2. All the substrategies from section 7.1 are synthesized into a global strategy $\hat{\mathcal{S}}_{ABC}$.

*Section* 8. The proofs built by $\hat{\mathcal{S}}_{ABC}$ are naturally structured as "proof-trees." We examine here a hypothetical infinite branch belonging to some infinite proof built by $\hat{\mathcal{S}}_{ABC}$. In section 8.2 we prove that such an infinite branch must possess an infinite suffix, which is a *B-stacking sequence*: roughly speaking, it begins with a $T_B$-application and later on, each time some $T_B^\alpha$ is applied, the vector which is used by $T_B^\alpha$ for constructing the new vector (on side $\alpha$) has a norm which is *large enough.*

We show carefully (Lemmas 8.4 to 8.10) that such a sequence of equations contains a subsequence of equations, on which the triangulation process (section 5) can be applied. Consequently the substrategy $T_C$ could apply to one node of this suffix.

*Section* 9. From the technical result proved in section 8, we quickly deduce that $\hat{\mathcal{S}}_{ABC}$ cannot build an *infinite* proof-tree from any true assertion. Unfortunately two problems remain to be solved:

- we must show that $\hat{\mathcal{S}}_{ABC}$ really builds a finite proof-tree from every true assertion, i.e., that $\hat{\mathcal{S}}_{ABC}$ is *closed*;
- at this point we do not know whether $\mathcal{B}_0$ is recursively enumerable or not, because of metarule (R5): this rule specifies that the conclusion is a pair of ... $\sigma$-$\eta$-bisimilar vectors, which is somewhat circular.

*Section* 10.

10.1. We overcome here the two difficulties quoted above:

- $\hat{\mathcal{S}}_{ABC}$ is shown to be *closed*;
- we show that every proof-tree constructed by $\hat{\mathcal{S}}_{ABC}$ is in fact also a proof for the smaller system $\mathcal{B}_1$ obtained from $\mathcal{B}_0$ by removing rule (R5) (proof of Theorem 10.6).

In other words we show that (R5) can be *eliminated* from $\mathcal{B}_0$, while preserving completeness. Hence $\mathcal{B}_1$ is a deduction system which is sound, complete, and recursively enumerable. The main decidability theorem (Theorem 10.7) follows: The bisimulation problem for rooted equational 1-graphs of finite out-degree is decidable.

10.2, 10.3. We perform successive simplifications of system $\mathcal{B}_1$. We finally obtain a system $\mathcal{B}_3$ whose elementary rules just express the basic algebraic properties of bisimulation: the fact that it is an equivalence relation, that it is compatible with right and left (matricial) products, that Arden's lemma remains true modulo bisimulation and, at last, its link with one-step derivation (rule (R34)).

We provide the reader with a roadmap which might help him in finding his way across the different sections (see Figure 1). The essential steps of the proof are placed in a column corresponding to the main theme to which they belong (Graphs, Algebra,

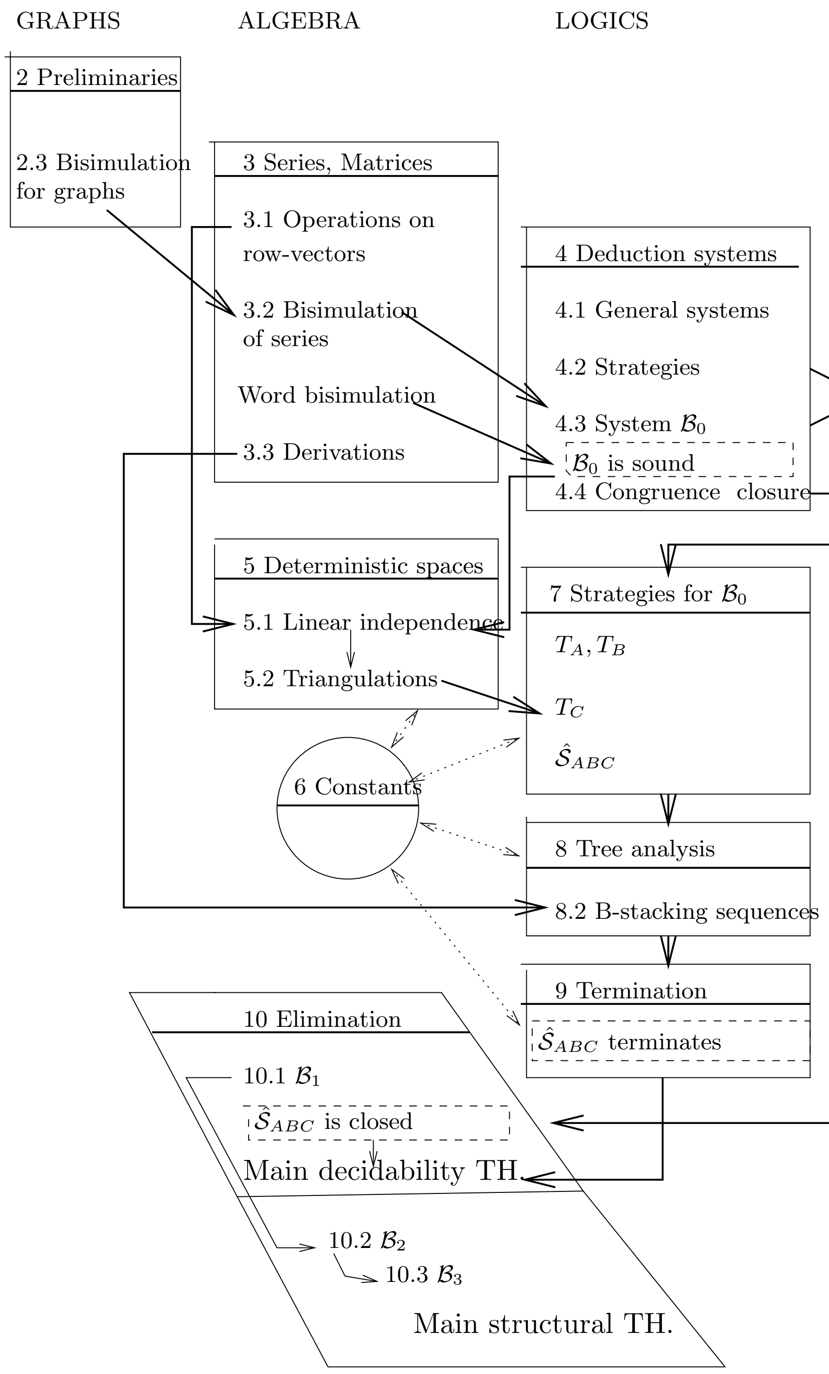


FIG. 1. *Roadmap.*

or Logics) and in a line corresponding to the physical linear ordering of the proof (from beginning to end). Arrows are added showing the main logical dependencies (they define a partial ordering over the steps that the written proof must extend into a total ordering).

**Overview: Small scale.** The proof exposed here is an updated version of the full proof given in [33] and presented concisely in [37]. This last reference can be used as a small-scale overview of our proof.

## 2. Preliminaries.

**2.1. Graphs.** Let $X$ be a finite alphabet. We call *graph over* $X$ any pair $\Gamma = (V_\Gamma, E_\Gamma)$ where $V_\Gamma$ is a set and $E_\Gamma$ is a subset of $V_\Gamma \times X \times V_\Gamma$. For every integer $n \in \mathbb{N}$, we call an $n$-graph every $n+2$-tuple $\Gamma = (V_\Gamma, E_\Gamma, v_1, \ldots, v_n)$ where $(V_\Gamma, E_\Gamma)$ is a graph and $(v_1, \ldots, v_n)$ is a sequence of distinguished vertices: they are called the *sources* of $\Gamma$.

A 1-graph $(V, E, v_1)$ is said to be *rooted* iff $v_1$ is a root of $(V, E)$ and $V \neq \{v_1\}$. A 2-graph $(V, E, v_1, v_2)$ is said *birooted* iff $v_1$ is a root, $v_2$ is a coroot of $(V, E)$, $v_1 \neq v_2$, and there is no edge going out of $v_2$ (this last technical condition will be useful for reducing the bisimilarity notion for graphs to an analogous notion on series; see sections 2.1, 2.3, and 3.2).

The *equational* graphs are the least solutions (in a suitable sense) of the systems of (hyperedge) graph-equations (see [12] for precise definitions). Let us mention that the equational graphs of *finite degree* are exactly the *context-free* graphs defined in [27].

**Bisimulations.**

DEFINITION 2.1. *Let $\Gamma = (V_\Gamma, E_\Gamma, v_1, \ldots, v_n)$, $\Gamma' = (V_{\Gamma'}, E_{\Gamma'}, v'_1, \ldots, v'_n)$ be two $n$-graphs over an alphabet $X$. Let $\mathcal{R}$ be some binary relation $\mathcal{R} \subseteq V_\Gamma \times V_{\Gamma'}$. $\mathcal{R}$ is a* simulation *from $\Gamma$ to $\Gamma'$ iff*

(1) $\mathrm{dom}(\mathcal{R}) = V_\Gamma$,

(2) $\forall i \in [1, n]$, $(v_i, v'_i) \in \mathcal{R}$,

(3) $\forall v \in V_\Gamma$, $w \in V_\Gamma$, $v' \in V_{\Gamma'}$, $x \in X$, *such that* $(v, x, w) \in E_\Gamma$ *and* $v\mathcal{R}v'$, *there exists* $w' \in V_{\Gamma'}$ *such that* $(v', x, w') \in E_{\Gamma'}$ *and* $w\mathcal{R}w'$.

*$\mathcal{R}$ is a* bisimulation *from $\Gamma$ to $\Gamma'$ iff $\mathcal{R}$ is a simulation from $\Gamma$ to $\Gamma'$ and $\mathcal{R}^{-1}$ is a simulation from $\Gamma'$ to $\Gamma$.*

This definition corresponds to the standard one [29, 26, 6] in the case where $n = 0$. The $n$-graphs $\Gamma, \Gamma'$ are said to be *bisimilar*, which we denote by $\Gamma \sim \Gamma'$, iff there exists a bisimulation $\mathcal{R}$ from $\Gamma$ to $\Gamma'$.

Let us now extend this definition by means of a relational morphism between free monoids.

DEFINITION 2.2. *Let $X, X'$ be two alphabets. A binary relation $\eta \subseteq X^* \times X'^*$ is called a strong relational morphism from $X^*$ to $X'^*$ iff*

(1) *$\eta$ is a submonoid of $X^* \times X'^*$,*

(2) $\mathrm{dom}(\eta) = X^*$, $\mathrm{im}(\eta) = X'^*$,

(3) *$\eta$ is generated (as a submonoid) by the subset $\eta \cap (X \times X')$.*

One can easily check that strong relational morphisms are preserved by inversion and composition and that any surjective map $\eta : X \to X'$ induces a strong relational morphism from $X^*$ to $X'^*$. Let $\Gamma = (V_\Gamma, E_\Gamma, v_1, \ldots, v_n)$ be an $n$-graph over the alphabet $X$, and let $\Gamma' = (V_{\Gamma'}, E_{\Gamma'}, v'_1, \ldots, v'_n)$ be an $n$-graph over the alphabet $X'$. Let $\eta \subseteq X^* \times X'^*$ be some strong relational morphism, and let $\mathcal{R}$ be some binary relation $\mathcal{R} \subseteq V_\Gamma \times V_{\Gamma'}$.

DEFINITION 2.3. *$\mathcal{R}$ is an $\eta$-*simulation *from $\Gamma$ to $\Gamma'$ iff*

(1) $\mathrm{dom}(\mathcal{R}) = V_\Gamma$,

(2) $\forall i \in [1, n]$, $(v_i, v'_i) \in \mathcal{R}$,

(3) $\forall v, w \in V_\Gamma$, $v' \in V_{\Gamma'}$, $x \in X$, *such that* $(v, x, w) \in E_\Gamma$ *and* $v\mathcal{R}v'$,

$$\exists w' \in V_{\Gamma'},\ x' \in \eta(x) \text{ such that } (v', x', w') \in E_{\Gamma'} \text{ and } w\mathcal{R}w'.$$

$\mathcal{R}$ *is an* $\eta$-bisimulation *iff* $\mathcal{R}$ *is an* $\eta$*-simulation and* $\mathcal{R}^{-1}$ *is an* $\eta^{-1}$*-simulation.*

For every $v \in V_\Gamma$, $v' \in V_{\Gamma'}$, we denote by $v \sim v'$ the fact that there exists some $\eta$-bisimulation $\mathcal{R}$ from $\Gamma$ to $\Gamma'$ such that $(v, v') \in \mathcal{R}$. Throughout this work, the composition of binary relations is denoted by $\circ$ and defined by the following: if $\mathcal{R}_1 \subseteq E \times F$ and $\mathcal{R}_2 \subseteq F \times G$, then

$$\mathcal{R}_1 \circ \mathcal{R}_2 = \{(x, z) \in E \times G \mid \exists y \in F, (x, y) \in \mathcal{R}_1, (y, z) \in \mathcal{R}_2\}. \tag{2.1}$$

FACT 2.4.
(1) *If* $\mathcal{R}$ *is an* $\eta$*-bisimulation, then* $\mathcal{R}^{-1}$ *is an* $\eta^{-1}$*-bisimulation.*
(2) *If* $\mathcal{R}_1$ *is an* $\eta_1$*-bisimulation and* $\mathcal{R}_2$ *is an* $\eta_2$*-bisimulation, then* $\mathcal{R}_1 \circ \mathcal{R}_2$ *is an* $\eta_1 \circ \eta_2$*-bisimulation.*
(3) *If for every* $i \in I$*,* $\mathcal{R}_i$ *is an* $\eta$*-bisimulation, then* $\bigcup_{i\in I} \mathcal{R}_i$ *is an* $\eta$*-bisimulation.*

**2.2. Pushdown automata.** A *pushdown automaton* (pda) on the alphabet $X$ is a 7-tuple $\mathcal{M} = \langle X, Z, Q, \delta, q_0, z_0, F\rangle$ where $Z$ is the finite stack-alphabet, $Q$ is the finite set of states, $q_0 \in Q$ is the initial state, $z_0$ is the initial stack-symbol, $F$ is a finite subset of $QZ^*$, the set of *final* configurations, and $\delta$, the transition function, is a mapping $\delta : QZ \times (X \cup \{\epsilon\}) \to \mathcal{P}_f(QZ^*)$.

Let $q, q' \in Q$, $\omega, \omega' \in Z^*$, $z \in Z$, $f \in X^*$, and $a \in X \cup \{\epsilon\}$; we note that $(qz\omega, af) \longmapsto_{\mathcal{M}} (q'\omega'\omega, f)$ if $q'\omega' \in \delta(qz, a)$. The binary relation $\overset{*}{\longmapsto}_{\mathcal{M}}$ is the reflexive and transitive closure of $\longmapsto_{\mathcal{M}}$.

For every $q\omega, q'\omega' \in QZ^*$ and $f \in X^*$, we note $q\omega \xrightarrow{f}_{\mathcal{M}} q'\omega'$ iff $(q\omega, f) \overset{*}{\longmapsto}_{\mathcal{M}} (q'\omega', \epsilon)$.

$\mathcal{M}$ is said to be *deterministic* iff, for every $z \in Z$, $q \in Q$, $x \in X$,

$$\mathrm{Card}(\delta(qz, \epsilon)) \in \{0, 1\}, \tag{2.2}$$

$$\mathrm{Card}(\delta(qz, \epsilon)) = 1 \Rightarrow \mathrm{Card}(\delta(qz, x)) = 0, \tag{2.3}$$

$$\mathrm{Card}(\delta(qz, \epsilon)) = 0 \Rightarrow \mathrm{Card}(\delta(qz, x)) \leq 1. \tag{2.4}$$

$\mathcal{M}$ is said to be *real-time* iff, for every $q \in Q$, $z \in Z$, $\mathrm{Card}(\delta(qz, \epsilon)) = 0$.

A configuration $q\omega$ of $\mathcal{M}$ is said to be $\epsilon$-bound iff there exists a configuration $q'\omega'$ such that $(q\omega, \epsilon) \longmapsto_{\mathcal{M}} (q'\omega', \epsilon)$; $q\omega$ is said to be $\epsilon$-free iff it is not $\epsilon$-bound.

A pda $\mathcal{M}$ is said to be *normalized* iff it fulfills conditions (2.2), (2.3) above and (2.5), (2.6), (2.7):

$$q_0 z_0 \text{ is } \epsilon\text{-free} \tag{2.5}$$

and for every $q \in Q$, $z \in Z$, $x \in X$,

$$q'\omega' \in \delta(qz, x) \Rightarrow \mid \omega' \mid \leq 2, \tag{2.6}$$

$$q'\omega' \in \delta(qz, \epsilon) \Rightarrow \mid \omega' \mid = 0. \tag{2.7}$$

All the pda considered here are assumed to fulfill condition (2.5). A pda $\mathcal{M}$ is called *birooted* iff it fulfills (2.8) and (2.9):

$$\exists \bar{q} \in Q, F = \{\bar{q}\}, \tag{2.8}$$

$$\forall q \in Q, \omega \in Z^*, f \in X^*, \quad q_0 z_0 \xrightarrow{f}_{\mathcal{M}} q\omega \Rightarrow \exists g \in X^*, q\omega \xrightarrow{g}_{\mathcal{M}} \bar{q}. \tag{2.9}$$

The *language recognized* by $\mathcal{M}$ is

$$\mathrm{L}(\mathcal{M}) = \{w \in X^* \mid \exists c \in F, q_0 z_0 \xrightarrow{w}_{\mathcal{M}} c\}.$$

It is a “folklore” result that, given a dpda $\mathcal{M}$, one can effectively compute another dpda $\mathcal{M}'$ which is normalized and fulfills

$$\mathrm{L}(\mathcal{M}') = \mathrm{L}(\mathcal{M}) - \{\varepsilon\}.$$

## 2.3. Graphs and pda.

**Equational graphs and pda.** We call a *transition-graph* of a pda $\mathcal{M}$, denoted $\mathcal{T}(\mathcal{M})$, the 0-graph $\mathcal{T}(\mathcal{M}) = (V_{\mathcal{T}(\mathcal{M})}, E_{\mathcal{T}(\mathcal{M})})$, where $V_{\mathcal{T}(\mathcal{M})} = \{q\omega \mid q \in Q,\ \omega \in Z^*,$ $q\omega$ is $\epsilon$-free$\}$ and

$$E_{\mathcal{T}(\mathcal{A})} = \{(c, x, c') \in V_{\mathcal{T}(\mathcal{M})} \times V_{\mathcal{T}(\mathcal{M})} \mid c \xrightarrow{x}_{\mathcal{M}} c'\}. \tag{2.10}$$

We call a *computation* 1-*graph* of the pda $\mathcal{M}$, denoted $(\mathcal{C}(\mathcal{M}), v_{\mathcal{M}})$, the subgraph of $\mathcal{T}(\mathcal{M})$ induced by the set of vertices which are accessible from the vertex $q_0 z_0$, together with the source $v_{\mathcal{M}} = q_0 z_0$. In the case where $\mathcal{M}$ is birooted, we call a *computation* 2-*graph* of the pda $\mathcal{M}$, denoted $(\mathcal{C}(\mathcal{M}), v_{\mathcal{M}}, \bar{v}_{\mathcal{M}})$, the graph $\mathcal{C}(\mathcal{M})$ defined just above, together with the sources $v_{\mathcal{M}} = q_0 z_0$, $\bar{v}_{\mathcal{M}} = \bar{q}$.

THEOREM 2.5. *Let $\Gamma = (\Gamma_0, v_0)$ be a rooted 1-graph over $X$. The following conditions are equivalent:*

(1) $\Gamma$ *is equational and has finite out-degree.*

(2) $\Gamma$ *is isomorphic to the computation 1-graph $(\mathcal{C}(\mathcal{M}), v_{\mathcal{M}})$ of some normalized pda $\mathcal{M}$.*

The formal proof of this theorem is quite technical and is omitted here. (See the appendix for a sketch of proof.)

COROLLARY 2.6. *Let $\Gamma = (\Gamma_0, v_0, \bar{v})$ be a birooted 2-graph over $X$. The following conditions are equivalent:*

(1) $\Gamma$ *is equational and has finite out-degree.*

(2) $\Gamma$ *is isomorphic to the computation 2-graph $(\mathcal{C}(\mathcal{M}), v_{\mathcal{M}}, \bar{v}_{\mathcal{M}})$ of some birooted normalized pda $\mathcal{M}$.*

**Bisimulation for nondeterministic (versus deterministic) graphs.** In this section, we reduce the classical notion of *bisimulation* for equational graphs to the notion of $\eta$-bisimulation for *deterministic* equational graphs, where $\eta$ has been suitably chosen (see Definition 2.3).

LEMMA 2.7. *Let $\Gamma_1$ be some rooted equational 1-graph over a finite alphabet $Y_1$ and let # be a new letter $\# \notin Y_1$. Then one can construct an equational birooted 2-graph $\Gamma$ over the alphabet $Y = Y_1 \cup \{\#\}$ such that*

(1) $V_{\Gamma_1} \subseteq V_\Gamma$,

(2) *for every $v, v' \in V_{\Gamma_1}$, ($v, v'$ are bisimilar in $\Gamma_1$) iff ($v, v'$ are bisimilar in $\Gamma$),*

(3) $\Gamma_1$ *has finite out-degree iff $\Gamma$ has finite out-degree.*

*Sketch of proof.* Let us define $\Gamma$ from $\Gamma_1$ by

$$V_\Gamma = V_{\Gamma_1} \cup \{\bar{v}\}, \quad E_\Gamma = E_{\Gamma_1} \cup \{(w, \#, \bar{v}) \mid w \in V_{\Gamma_1}\}, \quad \Gamma = (\Gamma_1, \bar{v}),$$

where $\bar{v}$ is a new vertex $\bar{v} \notin V_{\Gamma_1}$. One can easily check that $\Gamma$ is equational iff $\Gamma_1$ is equational and that, provided $\Gamma_1$ is rooted, $\Gamma$ is birooted. Points (1) and (3) of the lemma are clear. One can check that the mapping $\mathcal{R} \mapsto \mathcal{R} \cup \{(\bar{v}, \bar{v})\}$ is a bijection from the set of all the bisimulations over $\Gamma_1$ (i.e., from $\Gamma_1$ to $\Gamma_1$) to the set of all the bisimulations over $\Gamma$. Hence point (2) is true. $\square$

Let us consider finite alphabets $X, Y$, a length-preserving homomorphism $\psi : X^* \to Y^*$, and the strong relational morphism $\bar{\psi} = \psi \circ \psi^{-1} \subseteq X^* \times X^*$. An $n$-graph $\Gamma$ over $X$ will be called *$\bar{\psi}$-saturated* iff, for every $v \in V_\Gamma$, for every $(x, x') \in \bar{\psi}$,

$$(\exists v_1 \in V_\Gamma, (v, x, v_1) \in E_\Gamma) \Leftrightarrow (\exists v_1' \in V_\Gamma, (v, x', v_1') \in E_\Gamma).$$

LEMMA 2.8. *Let $\Gamma_1$ be an equational birooted 2-graph of finite out-degree over an alphabet $Y$. One can construct a finite alphabet $X$, a surjective length-preserving homomorphism $\psi : X^* \to Y^*$, and an equational, birooted 2-graph $\Gamma$ over the alphabet $X$, such that*

(1) $\Gamma$ *is deterministic,*
(2) $\Gamma$ *is $\bar{\psi}$-saturated,*
(3) $V_{\Gamma_1} = V_\Gamma$,
(4) $\mathrm{Id} : V_\Gamma \to V_{\Gamma_1}$ *is a $\psi$-bisimulation from $\Gamma$ to $\Gamma_1$.*

*Sketch of proof.* By Lemma 2.6, we can suppose that $\Gamma_1$ is the computation 2-graph $(\mathcal{C}(\mathcal{M}_1), v_{\mathcal{M}_1}, \bar{v}_{\mathcal{M}_1})$ of some birooted normalized pda $\mathcal{M}_1 = \langle Y, Z, Q, \delta_1, q_0, z_0, \{\bar{q}\}\rangle$. Let us consider the following integers: $\forall q \in Q$, $z \in Z$, $y \in Y$,

$$t_1(qz, y) = \mathrm{Card}(\delta_1(qz, y)), \quad \bar{t_1} = \max\{t_1(qz, y) \mid q \in Q, z \in Z, y \in Y\}.$$

Let $X = Y \times [1, \bar{t_1}]$ and let $\psi : X \to Y$ be the first projection. Let $\rho : QZ \times Y \times \mathbb{N} \to QZ^*$ such that $\mathrm{dom}(\rho) = \bigcup_{q \in Q, z \in Z, y \in Y} \{qz\} \times \{y\} \times [1, t_1(qz, y)]$ and

$$\rho(qz, y, \star) : \{qz\} \times \{y\} \times [1, t_1(qz, y)] \to \delta_1(qz, y)$$

is a bijection (for every triple $(q, z, y)$). We then define $\mathcal{M} = \langle X, Z, Q, \delta, q_0, z_0, \{\bar{q}\}\rangle$ by the following: for every $q \in Q$, $z \in Z$, $y \in Y$, $i \in [1, \bar{t_1}]$,

$$\delta(qz, \epsilon) = \delta_1(qz, \epsilon) \text{ if } qz \text{ is } \epsilon\text{-bound},$$

$$\delta(qz, (y, i)) = \{q'\omega'\} \text{ if}$$
$$\rho(qz, y, i) = q'\omega' \text{ or } [(1 \le t_1(qz, y) < i \le \bar{t_1} \text{ and } \rho(qz, y, 1) = q'\omega')].$$

The 2-graph $\Gamma = (\mathcal{C}(\mathcal{M}), v_\mathcal{M}, \bar{v}_\mathcal{M})$ fulfills the required properties. $\square$

Let us remark that, by point (4) and by composition of $\eta$-bisimulations, for every $v, v' \in V_\Gamma$, $v, v'$ are $\bar{\psi}$-bisimilar (w.r.t. $\Gamma$) iff $v, v'$ are bisimilar (w.r.t. $\Gamma_1$).

**2.4. Deterministic context-free grammars.** Let $\mathcal{M}$ be some dpda (we suppose here that $\mathcal{M}$ is normalized). The *variable* alphabet $V_\mathcal{M}$ associated to $\mathcal{M}$ is defined as

$$V_\mathcal{M} = \{[p, z, q] \mid p, q \in Q, z \in Z\}.$$

The *context-free* grammar $G_\mathcal{M}$ associated to $\mathcal{M}$ is then

$$G_\mathcal{M} = \langle X, V_\mathcal{M}, P_\mathcal{M}\rangle,$$

where $P_\mathcal{M}$ is the set of all the pairs of one of the following forms:

$$([p,z,q], x[p',z_1,p''][p'',z_2,q]), \tag{2.11}$$

where $p,q,p',p'' \in Q$, $x \in X$, $p'z_1z_2 \in \delta(pz,x)$,

$$([p,z,q], x[p',z',q]), \tag{2.12}$$

where $p,q,p' \in Q$, $x \in X$, $p'z' \in \delta(pz,x)$,

$$([p,z,q], a), \tag{2.13}$$

where $p,q, \in Q$, $a \in X \cup \{\epsilon\}$, $q \in \delta(pz,a)$. $G_{\mathcal{M}}$ is a *strict-deterministic* grammar (see Definition 3.7 below). A general theory of this class of grammars is presented in [15] and used in [16].

**2.5. Free monoids acting on semirings.**

**Semiring $\mathsf{B}\langle\langle\, W \,\rangle\rangle$.** Let $(\mathsf{B},+,\cdot,0,1)$, where $\mathsf{B}=\{0,1\}$ denotes the semiring of "booleans." Let $W$ be some alphabet. By $(\mathsf{B}\langle\langle\, W \,\rangle\rangle,+,\cdot,\emptyset,\epsilon)$ we denote the semiring of *boolean series* over $W$: the set $\mathsf{B}\langle\langle\, W \,\rangle\rangle$ is defined as $\mathsf{B}^{W^*}$; the sum and product are defined as usual; each word $w \in W^*$ can be identified with the element of $\mathsf{B}^{W^*}$ mapping the word $w$ on 1 and every other word $w' \neq w$ on 0; every boolean series $S \in \mathsf{B}\langle\langle\, W \,\rangle\rangle$ can then be written in a unique way as

$$S = \sum_{w \in W^*} S_w \cdot w,$$

where, for every $w \in W^*$, $S_w \in \mathsf{B}$.

The *support* of $S$ is the language

$$\mathrm{supp}(S) = \{w \in W^* \mid S_w \neq 0\}.$$

In the particular case where the semiring of coefficients is $\mathsf{B}$ (which is the only case considered in this article) we sometimes identify the series $S$ with its support. A series $S \in \mathsf{B}\langle\langle\, W \,\rangle\rangle$ is called a boolean *polynomial* over $W$ iff its support is *finite.* The set of all boolean polynomials over $W$ is denoted by $\mathsf{B}\langle W\rangle$.

The usual ordering $\leq$ on $\mathsf{B}$ extends to $\mathsf{B}\langle\langle\, W \,\rangle\rangle$ by

$$S \leq S' \text{ iff } \forall w \in W^*,\, S_w \leq S'_w.$$

We recall that for every $S \in \mathsf{B}\langle\langle\, W \,\rangle\rangle$, $S^*$ is the series defined by

$$S^* = \sum_{0 \leq n} S^n. \tag{2.14}$$

Given two alphabets $W, W'$, a map $\psi : \mathsf{B}\langle\langle\, W \,\rangle\rangle \to \mathsf{B}\langle\langle\, W' \,\rangle\rangle$ is said to be *$\sigma$-additive* iff it fulfills the following condition: for every denumerable family $(S_i)_{i\in\mathbb{N}}$ of elements of $\mathsf{B}\langle\langle\, W \,\rangle\rangle$,

$$\psi\left(\sum_{i\in\mathbb{N}} S_i\right) = \sum_{i\in\mathbb{N}} \psi(S_i). \tag{2.15}$$

A map $\psi : \mathsf{B}\langle\langle\, W \,\rangle\rangle \to \mathsf{B}\langle\langle\, W' \,\rangle\rangle$ which is both a semiring homomorphism and a $\sigma$-additive map is usually called a *substitution.*

**Actions of monoids.** Given a semiring $(\mathsf{S},+,\cdot,0,1)$ and a monoid $(\mathsf{M},\cdot,1_M)$, a map $\circ : \mathsf{S}\times\mathsf{M}\to\mathsf{S}$ is called a *right-action* of the monoid $\mathsf{M}$ over the semiring $\mathsf{S}$ iff, for every $S,T\in\mathsf{S}$, $m,m'\in\mathsf{M}$,

$$\begin{gathered} 0\circ m = 0, \quad S\circ 1_M = S, \\ (S+T)\circ m = (S\circ m)+(T\circ m), \quad S\circ(m\cdot m') = (S\circ m)\circ m'. \end{gathered} \tag{2.16}$$

In the particular case where $\mathsf{S}=\mathsf{B}\langle\langle\, W\,\rangle\rangle$, $\circ$ is said to be a $\sigma$-right-action if it fulfills the additional property that, for every denumerable family $(S_i)_{i\in\mathbb{N}}$ of elements of $\mathsf{S}$ and $m\in\mathsf{M}$,

$$\left(\sum_{i\in\mathbb{N}} S_i\right)\circ m = \sum_{i\in\mathbb{N}}(S_i\circ m). \tag{2.17}$$

**The action of $W^*$ on $\mathsf{B}\langle\langle\, W\,\rangle\rangle$.** We recall the following classical $\sigma$-right-action $\bullet$ of the monoid $W^*$ over the semiring $\mathsf{B}\langle\langle\, W\,\rangle\rangle$: $\forall S,S'\in\mathsf{B}\langle\langle\, W\,\rangle\rangle$, $u\in W^*$,

$$S\bullet u = S' \Leftrightarrow \forall w\in W^*, (S'_w = S_{u\cdot w})$$

(i.e., $S\bullet u$ is the *left-quotient* of $S$ by $u$, or the *residual* of $S$ by $u$).

For every $S\in\mathsf{B}\langle\langle\, W\,\rangle\rangle$ we denote by $\mathsf{Q}(S)$ the set of residuals of $S$:

$$\mathsf{Q}(S) = \{S\bullet u \mid u\in W^*\}.$$

We recall that $S$ is said to be *rational* iff the set $\mathsf{Q}(S)$ is *finite*. We define the *norm* of a series $S\in\mathsf{B}\langle\langle\, W\,\rangle\rangle$, denoted $\|S\|$, by

$$\|S\| = \mathrm{Card}(\mathsf{Q}(S))\in\mathbb{N}\cup\{\infty\}.$$

**The reduced grammar $G$.** The classical reduced and $\epsilon$-free grammar associated with $G_{\mathcal{M}}$ is $G_0=\langle X,V_0,P_0\rangle$, where

$$V_0 = \{v\in V_{\mathcal{M}} \mid \exists w\in X^+, v\overset{*}{\longrightarrow}_{P_{\mathcal{M}}} w\}, \tag{2.18}$$

$$\varphi_0 : \mathsf{B}\langle\langle\, V\,\rangle\rangle \to \mathsf{B}\langle\langle\, V_0\,\rangle\rangle$$

is the unique substitution such that, for every $v\in V$,

$$\varphi_0(v)=v \text{ (if } v\in V_0), \quad \varphi_0(v)=\epsilon \text{ (if } v\overset{*}{\longrightarrow}_{P_{\mathcal{M}}}\epsilon), \quad \varphi_0(v)=\emptyset \text{ (otherwise)},$$

$$\begin{aligned} P_0 = \{&(v,w')\in V_0\times(X\cup V_0)^+ \mid \\ &v\in V_0, \exists w\in(X\cup V_{\mathcal{M}})^*, (v,w)\in P_{\mathcal{M}}, w'=\varphi_0(w)\}. \end{aligned} \tag{2.19}$$

$G_0$ is the *reduced* and *$\epsilon$-free* form of $G_{\mathcal{M}}$. It is well known that, $\forall v\in V_0$,

$$\exists w\in X^+, v\overset{*}{\longrightarrow}_{P_0} w,$$

$$\{w\in X^*, v\overset{*}{\longrightarrow}_{P_{\mathcal{M}}} w\} = \{w\in X^*, v\overset{*}{\longrightarrow}_{P_0} w\}.$$

For technical reasons (which will be made clear in section 7), we introduce an alphabet of "marked variables" $\bar{V}_0$ together with a fixed bijection: $v\mapsto\bar{v}$ from $V_0$ to $\bar{V}_0$. Let

$V = V_0 \cup \bar{V}_0$. We denote by $\rho_e$ (the letter $e$ stands here for "erasing the marks") the literal morphism $V^* \to V_0^*$ defined by the following: for every $v \in V_0$,

$$\rho_e(v) = v, \quad \rho_e(\bar{v}) = v.$$

Similarly, $\bar{\rho}_e$ is the literal morphism $V^* \to \bar{V}_0^*$ defined by the following: for every $v \in V_0$,

$$\bar{\rho}_e(v) = \bar{v}, \quad \bar{\rho}_e(\bar{v}) = \bar{v}.$$

We denote also by $\rho_e, \bar{\rho}_e$ the unique substitutions extending these monoid homomorphisms.

At last, the grammar $G$ is defined by $G = \langle X, V, P \rangle$, where

$$P = P_0 \cup \{(\bar{\rho}_e(v), \bar{\rho}_e(w)) \mid (v, w) \in P_0\}.$$

In other words, the rules of $G$ consist of the rules of the usual proper and reduced grammar associated with $\mathcal{M}$ together with their marked copies.

**The action of $X^*$ on $\mathsf{B}\langle\langle\ V\ \rangle\rangle$.** Let us now fix a deterministic (normalized) pda $\mathcal{M}$ and consider the associated grammar $G$. We define a $\sigma$-right-action $\odot$ of the monoid $X^*$ over the semiring $\mathsf{B}\langle\langle\ V\ \rangle\rangle$ by the following: for every $v \in V$, $\beta \in V^*$, $x \in X$,

$$(v \cdot \beta) \odot x = \left( \sum_{(v,h) \in P} h \bullet x \right) \cdot \beta, \tag{2.20}$$

$$\epsilon \odot x = \emptyset. \tag{2.21}$$

Let us consider the unique substitution $\varphi : \mathsf{B}\langle\langle\ V\ \rangle\rangle \to \mathsf{B}\langle\langle\ X\ \rangle\rangle$ fulfilling the following: for every $v \in V$,

$$\varphi(v) = \{u \in X^* \mid v \overset{*}{\longrightarrow}_P u\}$$

(in other words, $\varphi$ maps every subset $L \subseteq V^*$ on the language generated by the grammar $G$ from the set of axioms $L$).

LEMMA 2.9. *For every $S \in \mathsf{B}\langle\langle\ V\ \rangle\rangle$, $u \in X^*$, $\varphi(S \odot u) = \varphi(S) \bullet u$ (i.e., $\varphi$ is a morphism of right-actions).*

*Proof.* Let $v \in V$, $\beta \in V^*$, $x \in X$. Recall that $G$ is in Greibach normal form (i.e., $P \subseteq V \times X \cdot V^*$). One can then check with formulas (2.20), (2.21) that

$$\varphi(\epsilon \odot x) = \varphi(\epsilon) \bullet x \quad \text{and} \quad \varphi((v \cdot \beta) \odot x) = \varphi(v \cdot \beta) \bullet x.$$

By induction on $|w|$, it follows that, $\forall w \in V^*$,

$$\varphi(w \odot x) = \varphi(w) \bullet x.$$

By $\sigma$-additivity of $\varphi$, it follows that, $\forall S \in \mathsf{B}\langle\langle\ V\ \rangle\rangle$,

$$\varphi(S \odot x) = \varphi(S) \bullet x.$$

By induction on $|u|$, it follows that, $\forall u \in X^*$,

$$\varphi(S \odot u) = \varphi(S) \bullet u. \qquad \square$$

We denote by $\equiv$ the kernel of $\varphi$, i.e., for every $S, T \in \mathsf{B}\langle\langle\ V\ \rangle\rangle$,

$$S \equiv T \Leftrightarrow \varphi(S) = \varphi(T).$$

## 3. Series and matrices.

**3.1. Deterministic series, vectors, and matrices.** We introduce here a notion of *deterministic* series which, in the case of the alphabet $V$ associated to a dpda $\mathcal{M}$, generalizes the classical notion of *configuration* of $\mathcal{M}$. The main advantage of this notion is that, unlike for configurations, we shall be able to define *nice algebraic operations* on these series (see, in particular, section 5.1). Let us consider a pair $(W, \smile)$ where $W$ is an alphabet and $\smile$ is an equivalence relation over $W$. We call $(W, \smile)$ a *structured* alphabet. We have the following two examples in mind:

- the case where $W = V_{\mathcal{M}}$, the variable alphabet associated to $\mathcal{M}$ and $[p, z, q] \smile [p', z', q']$ iff $p = p'$ and $z = z'$ (see [15]);
- the case where $W = X$, the terminal alphabet of $\mathcal{M}$ and $x \smile y$ holds for every $x, y \in X$ (see [15]).

**Definitions.**

DEFINITION 3.1. *Let $S \in \mathsf{B}\langle\langle\ W\ \rangle\rangle$. $S$ is said to be* left-deterministic *iff either*

(1) $S = \emptyset$ *or*
(2) $S = \epsilon$ *or*
(3) $\exists i_0 \in [1, m]$, $S_{i_0} \neq \emptyset$ *and* $\forall w, w' \in W^*$, $S_w = S_{w'} = 1 \Rightarrow [\exists A, A' \in W$, $w_1, w'_1 \in W^*$, $A \smile A'$, $w = A \cdot w_1$, *and* $w' = A' \cdot w'_1]$.

A left-deterministic series $S$ is said to have the type $\emptyset$ (resp., $\epsilon$, $[A]_\smile$) if case (1) (resp., (2), (3)) occurs.

DEFINITION 3.2. *Let $S \in \mathsf{B}\langle\langle\ W\ \rangle\rangle$. $S$ is said to be* deterministic *iff, for every $u \in W^*$, $S \bullet u$ is left-deterministic.*

This notion is the straightforward extension to the infinite case of the notion of a (finite) *set of associates* defined in [16, Definition 3.2, p. 188].

We denote by $\mathsf{DB}\langle\langle\ W\ \rangle\rangle$ the subset of deterministic boolean series over $W$. Let us denote by $\mathsf{B}_{n,m}\langle\langle\ W\ \rangle\rangle$ the set of $(n, m)$-matrices with entries in the semiring $\mathsf{B}\langle\langle\ W\ \rangle\rangle$.

DEFINITION 3.3. *Let $m \in \mathbb{N}$, $S \in \mathsf{B}_{1,m}\langle\langle\ W\ \rangle\rangle : S = (S_1, \ldots, S_m)$. $S$ is said to be* left-deterministic *iff either*

(1) $\forall i \in [1, m]$, $S_i = \emptyset$ *or*
(2) $\exists i_0 \in [1, m]$, $S_{i_0} = \epsilon$ *and* $\forall i \neq i_0$, $S_i = \emptyset$ *or*
(3) $\forall w, w' \in W^*$, $\forall i, j \in [1, m]$, $(S_i)_w = (S_j)_{w'} = 1 \Rightarrow [\exists A, A' \in W$, $w_1, w'_1 \in V^*$, $A \smile A'$, $w = A \cdot w_1$, *and* $w' = A' \cdot w'_1]$.

A left-deterministic row-vector $S$ is said to have the type $\emptyset$ (resp., $(\epsilon, i_0)$, $[A]_\smile$) if case (1) (resp., (2), (3)) occurs.

The right-action $\bullet$ on $\mathsf{B}\langle\langle\ W\ \rangle\rangle$ is extended componentwise to $\mathsf{B}_{n,m}\langle\langle\ W\ \rangle\rangle$: for every $S = (s_{i,j})$, $u \in W^*$, the matrix $T = S \bullet u$ is defined by

$$t_{i,j} = s_{i,j} \bullet u.$$

The ordering $\leq$ on $\mathsf{B}$ is also extended componentwise to $\mathsf{B}_{n,m}\langle\langle\ W\ \rangle\rangle$.

DEFINITION 3.4. *Let $S \in \mathsf{B}_{1,m}\langle\langle\ W\ \rangle\rangle$. $S$ is said to be* deterministic *iff, for every $u \in W^*$, $S \bullet u$ is left-deterministic.*

We denote by $\mathsf{DB}_{1,m}\langle\langle\ W\ \rangle\rangle$ the subset of deterministic row-vectors of dimension $m$ over $\mathsf{B}\langle\langle\ W\ \rangle\rangle$.

DEFINITION 3.5. *Let $S \in \mathsf{B}_{n,m}\langle\langle\ W\ \rangle\rangle$. $S$ is said to be* deterministic *iff, for every $i \in [1, n]$, $S_{i,\cdot}$ is a deterministic row-vector.*

Let us note first some easy facts about deterministic matrices.

FACT 3.6. *Let $S \in \mathsf{DB}\langle\langle\, W\,\rangle\rangle$. For every $T \in \mathsf{B}\langle\langle\, W\,\rangle\rangle$, $u \in W^*$*
(1) $T \leq S \Rightarrow T \in \mathsf{DB}\langle\langle\, W\,\rangle\rangle$.
(2) $T = S \bullet u \Rightarrow T \in \mathsf{DB}\langle\langle\, W\,\rangle\rangle$.

**Norm.** Let us generalize the classical definition of *rationality* of series in $\mathsf{B}\langle\langle\, W\,\rangle\rangle$ to matrices. Given $M \in \mathsf{B}_{n,m}\langle\langle\, W\,\rangle\rangle$ we denote by $\mathsf{Q}(M)$ the set of *residuals* of $M$:

$$\mathsf{Q}(M) = \{M \bullet u \mid u \in W^*\}.$$

Similarly, we denote by $\mathsf{Q}_r(M)$ the set of *row-residuals* of $M$:

$$\mathsf{Q}_r(M) = \bigcup_{1\leq i\leq n} \mathsf{Q}(M_{i,*}).$$

$M$ is said to be *rational* iff the set $\mathsf{Q}(M)$ is finite. One can check that it is equivalent to the property that every coefficient $M_{i,j}$ is rational, or to the property that $\mathsf{Q}_r(M)$ is finite. We denote by $\mathsf{RB}_{n,m}\langle\langle\, W\,\rangle\rangle$ (resp., $\mathsf{DRB}_{n,m}\langle\langle\, W\,\rangle\rangle$) the set of rational (resp., deterministic, rational) matrices over $\mathsf{B}\langle\langle\, W\,\rangle\rangle$. For every $M \in \mathsf{RB}_{n,m}\langle\langle\, W\,\rangle\rangle$, we define the norm of $M$ as

$$\|M\| = \mathrm{Card}(\mathsf{Q}_r(M)).$$

**Grammars.**
DEFINITION 3.7. *Let $G = \langle X, V, P\rangle$ be a context-free grammar in Greibach normal form. $G$ is said to be* strict-deterministic *iff there exists an equivalence relation $\smile$ over $V$ fulfilling the following condition: for every $E \in V$, $x \in X$, if $(E_k)_{1\leq k\leq m}$ is a bijection $[1,m] \to [E]_\smile$, and $H_k = \sum_{(E_k,h)\in P} h \bullet x$, then*

$$(H_1, H_2, \ldots, H_m) \text{ is a deterministic vector.}$$

*Any equivalence $\smile$ satisfying the above condition is said to be a* strict equivalence *for the grammar $G$.*

This definition is a reformulation of [15, Definition 11.4.1, p. 347] adapted to the case of a Greibach normal form.

THEOREM 3.8. *Let $G_1 = \langle X, V_1, P_1\rangle$ be a strict-deterministic grammar. Then its reduced form $G_0 = \langle X, V_0, P_0\rangle$, as defined in formulas* (2.18), (2.19), *is strict-deterministic too. Moreover, if $\smile$ is a strict equivalence for $G_1$, its restriction over $V_0$ is a strict equivalence for $G_0$.*

The proof would consist in slightly extending the proof of [15, Theorem 11.4.1, p. 350].

It is known that, given a dpda $\mathcal{M}$, its associated grammar $G_{\mathcal{M}}$ is strict-deterministic. By Theorem 3.8 $G_0$ is strict-deterministic too. Let us consider the minimal strict equivalence $\smile$ for $G_0$ and extend it to $V$ by, $\forall v, v' \in V_0$,

$$\bar{v} \smile \bar{v}' \Leftrightarrow v \smile v'; \quad \bar{v} \not\smile v'.$$

Then $\smile$ is a strict equivalence for $G$ (the grammar $G$ is defined in section 2.5). This ensures that $G$ is strict-deterministic.

**Residuals.**
LEMMA 3.9 (see [42, Lemma 37]). *Let $S \in \mathsf{DB}\langle\langle\, W\,\rangle\rangle$, $T \in \mathsf{B}\langle\langle\, W\,\rangle\rangle$, $u \in W^*$. If $S \bullet u \neq \emptyset$, then $(S \cdot T) \bullet u = (S \bullet u) \cdot T$.*

LEMMA 3.10 (see [42, Lemma 39]). *Let $S \in \mathsf{DB}_{1,m}\langle\langle\, W\,\rangle\rangle$, $T \in \mathsf{B}_{m,s}\langle\langle\, W\,\rangle\rangle$, $u \in W^*$, and $U = S \cdot T$. Exactly one of the following cases is true:*

(1) $\exists j$, $S_j \bullet u \notin \{\emptyset, \epsilon\}$. *In this case* $U \bullet u = (S \bullet u) \cdot T$.
(2) $\exists j_0$, $\exists u', u''$, $u = u' \cdot u''$, $S_{j_0} \bullet u' = \epsilon$. *In this case* $U \bullet u = T_{j_0} \bullet u''$.
(3) $\forall j$, $\forall u' \preceq u$, $S_j \bullet u = \emptyset$, $S_j \bullet u' \neq \epsilon$. *In this case* $U \bullet u = \emptyset = (S \bullet u) \cdot T$.

LEMMA 3.11. *For every* $S \in \mathsf{DB}_{n,m}\langle\langle\ W\ \rangle\rangle$, $T \in \mathsf{DB}_{m,s}\langle\langle\ W\ \rangle\rangle$, $S \cdot T \in \mathsf{DB}_{n,s}\langle\langle\ W\ \rangle\rangle$.

LEMMA 3.12. *Let* $A \in \mathsf{DB}_{n,m}\langle\langle\ W\ \rangle\rangle$, $B \in \mathsf{B}_{m,s}\langle\langle\ W\ \rangle\rangle$. *Then* $\|A \cdot B\| \leq \|A\| + \|B\|$.

The two above lemmas correspond to [42, Lemma 310].

***W* = *V*.** Let $(W, \smile)$ be the structured alphabet $(V, \smile)$ associated with $\mathcal{M}$ and let us consider a bijective numbering of the elements of $Q$: $(q_1, q_2, \dots, q_{n_Q})$. Let us define here a handful of notations for some particular vectors or matrices. Let us use the *Kronecker symbol* $\delta_{i,j}$ to mean $\epsilon$ if $i = j$ and $\emptyset$ if $i \neq j$. For every $1 \leq n$, $1 \leq i \leq n$, we define the row-vector $\epsilon_i^n$ as

$$\epsilon_i^n = (\epsilon_{i,j}^n)_{1 \leq j \leq n}, \quad \text{where } \forall j,\ \epsilon_{i,j}^n = \delta_{i,j}.$$

We call any vector of the form $\epsilon_i^n$ a *unit row-vector.*

For every $1 \leq n$, we denote by $\emptyset^n \in \mathsf{DB}_{1,n}\langle\langle\ V\ \rangle\rangle$ the row-vector

$$\emptyset^n = (\emptyset, \dots, \emptyset).$$

For every $\omega \in Z^*$, $p, q \in Q$, $[p\omega q]$ is the deterministic series defined inductively by

$$[p\epsilon q] = \emptyset \text{ if } p \neq q, \quad [p\epsilon q] = \epsilon \text{ if } p = q,$$

$$[p\omega q] = \sum_{r \in Q} [p, z, r] \cdot [r\omega' q] \text{ if } \omega = z \cdot \omega' \text{ for some } z \in Z,\ \omega' \in Z^*.$$

Let us define

$$K_0 = \max\{\|(E_1, E_2, \dots, E_n) \odot x\| \mid (E_i)_{1 \leq i \leq n} \text{ is a bijective numbering of some class in } V/\smile,\ x \in X\}. \tag{3.1}$$

LEMMA 3.13 (see [42, Lemma 318]). *For every* $S \in \mathsf{DB}_{1,\lambda}\langle\langle\ V\ \rangle\rangle$, $u \in X^*$,
(1) $S \odot u \in \mathsf{DB}_{1,\lambda}\langle\langle\ V\ \rangle\rangle$,
(2) $\|S \odot u\| \leq \|S\| + K_0 \cdot |u|$.

LEMMA 3.14 (see [42, Lemma 319]). *Let* $\lambda \in \mathbb{N} - \{0\}$, $S \in \mathsf{DRB}_{1,\lambda}\langle\langle\ V\ \rangle\rangle$, $u \in X^*$. *One of the three following cases must occur:*
(1) $S \odot u = \emptyset^\lambda$.
(2) $S \odot u = \epsilon_j^\lambda$ *for some* $j \in [1, \lambda]$.
(3) $\exists u_1, u_2 \in X^*$, $v_1 \in V^*$, $q \in \mathbb{N}$, $E_1, \dots, E_k, \dots, E_q \in V$, $\Phi \in \mathsf{DRB}_{q,\lambda}\langle\langle\ V\ \rangle\rangle$ *such that*

$$u = u_1 \cdot u_2,\ S \odot u_1 = S \bullet v_1 = \sum_{k=1}^{q} E_k \cdot \Phi_k,\ S \odot u = \sum_{k=1}^{q} (E_k \odot u_2) \cdot \Phi_k, \text{ and}$$

$$\forall k \in [1, q], E_k \smile E_1, E_k \odot u_2 \notin \{\epsilon, \emptyset\}.$$

We now give an adaptation of Lemma 3.10 to the action $\odot$ in place of $\bullet$.

LEMMA 3.15 (see [42, Lemma 321]). *Let* $S \in \mathsf{DB}_{1,m}\langle\langle\ V\ \rangle\rangle$, $T \in \mathsf{B}_{m,s}\langle\langle\ V\ \rangle\rangle$, $u \in X^*$, *and* $U = S \cdot T$. *Exactly one of the following cases is true:*
(1) $S \odot u \notin \{\emptyset^m\} \cup \{\epsilon_j^m \mid 1 \leq j \leq m\}$. *In this case* $U \odot u = (S \odot u) \cdot T$.
(2) $\exists j_0$, $\exists u', u''$, $u = u' \cdot u''$, $S \odot u' = \epsilon_{j_0}^s$. *In this case* $U \odot u = T_{j_0} \odot u''$.
(3) $\forall j$, $\forall u' \preceq u$, $S \odot u = \emptyset^m$ *and* $S \odot u' \neq \epsilon_j^m$. *In this case* $U \odot u = \emptyset^s = (S \odot u) \cdot T$.

**Marks.** A word $w \in V^*$ is said to be *marked* iff $w \in V^* \cdot \bar{V}_0 \cdot V^*$; it is said to be *fully marked* iff $w \in \bar{V}_0^*$.

A series $S \in \mathsf{B}\langle\langle\ V\ \rangle\rangle$ is said to be *marked* iff $\exists w \in \mathrm{supp}(S)$, $w$ is marked; it is said to be *fully marked* iff $\forall w \in \mathrm{supp}(S)$, $w$ is fully marked. It is said to be *unmarked* iff it is *not* marked. A matrix $S \in \mathsf{B}_{m,n}\langle\langle\ V\ \rangle\rangle$ is said to be marked (resp., fully marked, unmarked) iff, for every $i \in [1, m]$, the series $\sum_{j=1}^{n} S_{i,j}$ is marked (resp., fully marked, unmarked).

Definition 3.16. *Let $d \in \mathbb{N}$. A vector $S \in \mathsf{DB}_{1,\lambda}\langle\langle\ V\ \rangle\rangle$ is said to be d-marked iff there exists $q \in \mathbb{N}$, $\alpha \in \mathsf{DRB}_{1,q}\langle V\rangle$, $\Phi \in \mathsf{DRB}_{q,\lambda}\langle\langle\ V\ \rangle\rangle$ such that*

$$S = \sum_{k=1}^{q} \alpha_k \cdot \Phi_k \text{ and } \|\alpha\| \le d,$$

*and $\Phi$ is unmarked.*

Lemma 3.17. *For every $S \in \mathsf{DB}_{1,\lambda}\langle\langle\ V\ \rangle\rangle$,*

(1) *$\rho_e(S) \in \mathsf{DB}_{1,\lambda}\langle\langle\ V\ \rangle\rangle$,*

(2) *$\|\rho_e(S)\| \le \|S\|$.*

*Sketch of proof.*

(1) Let us notice that the homomorphism $\rho_e : V^* \to V^*$ preserves the equivalence $\smile$: for every $v, v' \in V$, if $v \smile v'$, then $\rho_e(v) \smile \rho_e(v')$. It follows that the corresponding substitution $\rho_e$ preserves determinism.

(2) Let $S \in \mathsf{DB}_{1,\lambda}\langle\langle\ V\ \rangle\rangle$. For every $v \in V_0$,

$$\rho_e(S) \bullet v = \rho_e(S \bullet v) \quad \text{or} \quad \rho_e(S) \bullet v = \rho_e(S \bullet \bar{v})$$

according to the fact that the leftmost letters of the monomials of $S$ are in $[v]_\smile$ or in $[\bar{v}]_\smile$; both formulas are true when $S$ is null or is a unit.

By induction on the length, it follows that, for every $w \in V_0^*$, there exists $w' \in V^*$ such that

$$\rho_e(w') = w \quad \text{and} \quad \rho_e(S) \bullet w = \rho_e(S \bullet w').$$

Moreover, for every $w \in V^* \bar{V}_0 V^*$,

$$\rho_e(S) \bullet w = \emptyset^\lambda,$$

but in this case, too, there exists some $w' \in V^*$ such that $\rho_e(S) \bullet w = \rho_e(S \bullet w')$.

The map $T \mapsto \rho_e(T)$ is then a surjective map from $\mathsf{Q}(S)$ onto $\mathsf{Q}(\rho_e(S))$, which proves that $\|\rho_e(S)\| \le \|S\|$. □

**Operations on row-vectors.** Let us introduce two new operations on row-vectors and prove some technical lemmas about them.

Given $A, B \in \mathsf{B}_{1,m}\langle\langle\ W\ \rangle\rangle$ and $1 \le j_0 \le m$ we define the vector $C = A \nabla_{j_0} B$ as follows: if $A = (a_1, \ldots, a_j, \ldots, a_m)$, $B = (b_1, \ldots, b_j, \ldots, b_m)$, then $C = (c_1, \ldots, c_j, \ldots, c_m)$, where

$$c_j = a_j + a_{j_0} \cdot b_j \text{ if } j \ne j_0, \quad c_j = \emptyset \text{ if } j = j_0.$$

Lemma 3.18 (see [42, Lemma 311]). *Let $A, B \in \mathsf{B}_{1,m}\langle\langle\ W\ \rangle\rangle$ and $1 \le j_0 \le m$.*

(1) *If $A, B$ are deterministic, then $A \nabla_{j_0} B$ is deterministic.*

(2) *If $A, B$ are deterministic, then $\|A \nabla_{j_0} B\| \le \|A\| + \|B\|$.*

Given $A \in \mathsf{DB}_{1,m}\langle\langle\ W\ \rangle\rangle$ and $1 \leq j_0 \leq m$ we define the vector $A' = \nabla^*_{j_0}(A)$ as follows: if $A = (a_1, \ldots, a_j, \ldots, a_m)$, then $A' = (a'_1, \ldots, a'_j, \ldots, a'_m)$, where

$$a'_j = a^*_{j_0} \cdot a_j \text{ if } j \neq j_0, \quad a'_j = \emptyset \text{ if } j = j_0.$$

Lemma 3.19 (see [42, Lemma 312]). *Let $A \in \mathsf{DB}_{1,m}\langle\langle\ W\ \rangle\rangle$ and $1 \leq j_0 \leq m$. Then $\nabla^*_{j_0}(A) \in \mathsf{DB}_{1,m}\langle\langle\ W\ \rangle\rangle$ and $\|\nabla^*_{j_0}(A)\| \leq \|A\|$.*

**3.2. Bisimulation of series.** Up to the end of this section, we consider the structured alphabet $V$ associated with a dpda $\mathcal{M}$ over $X$. We suppose a strong relational morphism $\eta \subseteq X^* \times X^*$ is given (see Definition 2.2).

**Series, words, and graphs.** Let us first give a slight adaptation of Definition 2.1 to the $n$-graph $(\mathsf{DRB}_{1,n}\langle\langle\ V\ \rangle\rangle, \odot, (\epsilon^n_i)_{1\leq i\leq n})$.

Definition 3.20. *Let $\mathcal{R}$ be some binary relation $\mathcal{R} \subseteq \mathsf{DRB}_{1,n}\langle\langle\ V\ \rangle\rangle \times \mathsf{DRB}_{1,n}\langle\langle\ V\ \rangle\rangle$. $\mathcal{R}$ is a $\sigma$-$\eta$-bisimulation iff*

(1) $\forall (S, S') \in \mathcal{R}$, $\forall x \in X$,

$$\exists x' \in \eta(x),\ (S \odot x, S' \odot x') \in \mathcal{R} \textit{ and } \exists x'' \in \eta^{-1}(x),\ (S \odot x'', S' \odot x) \in \mathcal{R},$$

(2) $\forall (S, S') \in \mathcal{R}$, $\forall i \in [1, n]$, $(S = \epsilon^n_i \Leftrightarrow S' = \epsilon^n_i)$.

We denote by $S \sim S'$ the fact that there exists some $\sigma$-$\eta$-bisimulation $\mathcal{R}$ such that $(S, S') \in \mathcal{R}$. One can notice that $\sim$ is the greatest $\sigma$-$\eta$-bisimulation (w.r.t. the inclusion ordering) over $\mathsf{DRB}_{1,n}\langle\langle\ V\ \rangle\rangle$. The $\sigma$-bisimulation relations can be conveniently expressed in terms of *word*-bisimulations.

Definition 3.21. *Let $S, S' \in \mathsf{DRB}_{1,n}\langle\langle\ V\ \rangle\rangle$ and $\mathcal{R} \subseteq X^* \times X^*$. $\mathcal{R}$ is a $w$-$\eta$-bisimulation with respect to $(S, S')$ iff $\mathcal{R} \subseteq \eta$ and*

(1) totality: $\mathrm{dom}(\mathcal{R}) = X^*$, $\mathrm{im}(\mathcal{R}) = X^*$*;*

(2) extension: $\forall (u, u') \in \mathcal{R}$, $\forall x \in X$,

$$\exists x' \in \eta(x),\ (u \cdot x, u' \cdot x') \in \mathcal{R} \textit{ and } \exists x'' \in \eta^{-1}(x),\ (u \cdot x'', u' \cdot x) \in \mathcal{R};$$

(3) coherence: $\forall (u, u') \in \mathcal{R}$, $\forall i \in [1, n]$, $(S \odot u = \epsilon^n_i) \Leftrightarrow (S' \odot u' = \epsilon^n_i)$*;*

(4) prefix: $\forall (u, u') \in X^* \times X^*$, $\forall (x, x') \in X \times X$, $(u \cdot x, u' \cdot x') \in \mathcal{R} \Rightarrow (u, u') \in \mathcal{R}$.

(Condition (1) can be equivalently replaced by "$(\epsilon, \epsilon) \in \mathcal{R}$.") $\mathcal{R}$ is said to be a $w$-$\eta$-bisimulation of *order* $m$ w.r.t. $(S, S')$ iff it fulfills conditions (3)–(4) above and the modified conditions

(1′) $\mathrm{dom}(\mathcal{R}) = X^{\leq m}$, $\mathrm{im}(\mathcal{R}) = X^{\leq m}$,

(2′) $\forall (u, u') \in \mathcal{R} \cap (X^{\leq m-1} \times X^{\leq m-1})$, $\forall x \in X$,

$$\exists x' \in \eta(x), (u \cdot x, u' \cdot x') \in \mathcal{R} \quad \text{and} \quad \exists x'' \in \eta^{-1}(x), (u \cdot x'', u' \cdot x) \in \mathcal{R}.$$

The $w$-$\eta$-bisimulations are also called $w$-$\eta$-bisimulations of *order* $\infty$. Lemmas 3.22 and 3.25 below relate the notions of $w$-$\eta$-bisimulation (on words), $\sigma$-$\eta$-bisimulation (on series), and $\eta$-bisimulation (on the vertices of the computation 2-graph of $\mathcal{M}$).

Lemma 3.22. *Let $S, S' \in \mathsf{DRB}_{1,n}\langle\langle\ V\ \rangle\rangle$. The following properties are equivalent:*

(i) *$S \sim S'$.*

(ii) *There exists $\mathcal{R} \subseteq X^* \times X^*$ which is a $w$-$\eta$-bisimulation w.r.t. $(S, S')$.*

(iii) *$\forall m \in \mathbb{N}$, there exists $\mathcal{R}_m \subseteq X^{\leq m} \times X^{\leq m}$ which is a $w$-$\eta$-bisimulation of order $m$ w.r.t. $(S, S')$.*

*Proof.* **(i) $\Rightarrow$ (iii):** Suppose that $\mathcal{S}$ is a $\sigma$-$\eta$-bisimulation w.r.t. $(S, S')$. Let us prove by induction on the integer $m$ the following property $\mathcal{P}(m)$: $\exists \mathcal{R}_m$, $w$-$\eta$-bisimulation of order $m$ w.r.t. $(S, S')$ such that

$$\forall (u, u') \in \mathcal{R}_m, \quad (S \odot u, S' \odot u') \in \mathcal{S}. \tag{3.2}$$

***m* = 0:** Let $\mathcal{R}_0 = \{(\epsilon, \epsilon)\}$. $\mathcal{R}_0$ clearly fulfills points (1′), (2′), (4) of the above definition. Moreover, as $(S, S') \in \mathcal{S}$, where $\mathcal{S}$ fulfills condition (2) of Definition 3.20, $\mathcal{R}_0$ fulfills point (3) of Definition 3.21.

**$m = m' + 1$:** Let $\mathcal{R}_{m'}$ be some $w$-$\eta$-bisimulation of order $m'$ w.r.t. $(S, S')$. Let us define $\mathcal{R}_m = \mathcal{R}_{m'} \cup \{(u \cdot x, u' \cdot x') \mid (u, u') \in \mathcal{R}_{m'}, (S \odot ux, S' \odot u'x') \in \mathcal{S}$, and $(x, x') \in \eta\}$. Property (1) of $\mathcal{S}$ and property (1′) of $\mathcal{R}_{m'}$ imply that

$$\mathrm{dom}(\mathcal{R}_m) = X^{\leq m}, \quad \mathrm{im}(\mathcal{R}_m) = X^{\leq m}. \tag{3.3}$$

Property (1) of $\mathcal{S}$ and property (2′) of $\mathcal{R}_{m'}$ imply that $\forall (u, u') \in \mathcal{R}_m \cap (X^{\leq m-1} \times X^{\leq m-1})$, $\forall x \in X$,

$$\exists x' \in \eta(x), (u \cdot x, u' \cdot x') \in \mathcal{R}_m \quad \text{and} \quad \exists x'' \in \eta^{-1}(x), (u \cdot x'', u' \cdot x) \in \mathcal{R}_m. \tag{3.4}$$

Property (2) of $\mathcal{S}$ and property (3) of $\mathcal{R}_{m'}$ imply that

$$\forall (u, u') \in \mathcal{R}_m, \forall i \in [1, n], \quad (S \odot u = \epsilon_i^n) \Leftrightarrow (S' \odot u' = \epsilon_i^n). \tag{3.5}$$

Property (4) of $\mathcal{R}_{m'}$ and the definition of $\mathcal{R}_m$ imply that

$$\forall (u, u') \in X^* \times X^*, \forall (x, x') \in X \times X, \quad (u \cdot x, u' \cdot x') \in \mathcal{R}_m \Rightarrow (u, u') \in \mathcal{R}_m. \tag{3.6}$$

Property (3.2) for $\mathcal{R}_{m'}$ and the definition of $\mathcal{R}_m$ imply that (3.2) is fulfilled by $\mathcal{R}_m$ too. Equations (3.3), (3.4), (3.5), (3.6) prove that $\mathcal{R}_m$ is a $w$-$\eta$-bisimulation of order $m$ w.r.t. $(S, S')$, and hence $\mathcal{P}(m)$ is proved.

**(iii) $\Rightarrow$ (ii):** Let us notice that, as the alphabet $X$ is finite, for every $w$-$\eta$-bisimulation $\mathcal{R}$ of order $m$ w.r.t. $(S, S')$,

$$\mathrm{Card}\{\mathcal{R}' \subseteq X^* \times X^* \mid \mathcal{R} \subseteq \mathcal{R}', \mathcal{R}' \text{ is a } w\text{-}\eta\text{-bisim. of ord. } m+1 \text{ w.r.t. } (S, S')\} < \infty.$$

Hence, by Koenig's lemma, if (iii) is true, then there exists an infinite sequence $(\mathcal{R}_m)_{m \in \mathbb{N}}$ such that for every $m \in \mathbb{N}$, $\mathcal{R}_m$ is a $w$-$\eta$-bisimulation of order $m$ w.r.t. $(S, S')$ and $\mathcal{R}_m \subseteq \mathcal{R}_{m+1}$. Let us then define

$$\mathcal{R} = \bigcup_{m \geq 0} \mathcal{R}_m.$$

$\mathcal{R}$ is a $w$-$\eta$-bisimulation of order $\infty$ w.r.t. $(S, S')$.

**(ii) $\Rightarrow$ (i):** Let $\mathcal{R}$ be a $w$-$\eta$-bisimulation of order $\infty$ w.r.t. $(S, S')$. Let us define a relation $\mathcal{S}$ by

$$\mathcal{S} = \{(S \odot u, S' \odot u') \mid (u, u') \in \mathcal{R}\}.$$

The totality property of $\mathcal{R}$ implies that $(S, S') \in \mathcal{S}$. The extension property of $\mathcal{R}$ implies that $\mathcal{S}$ fulfills condition (1) of Definition 3.20 and the coherence property of $\mathcal{R}$ implies that $\mathcal{S}$ fulfills condition (2). $\square$

Lemma 3.22 leads naturally to the following definition. We denote by $\mathcal{B}_n(T, T')$ the set of all $w$-$\eta$-bisimulations of order $n$ w.r.t. $(T, T')$.

DEFINITION 3.23. *Let $\lambda \in \mathbb{N} - \{0\}$, $S, S' \in \mathsf{DRB}_{1,\lambda}\langle\langle\ V\ \rangle\rangle$. We define the* divergence *between $S$ and $S'$ as*

$$\mathrm{Div}(S, S') = \inf\{n \in \mathbb{N} \mid \mathcal{B}_n(S, S') = \emptyset\}.$$

*(It is understood that* $\inf(\emptyset) = \infty$.*)*

Let us suppose that the dpda $\mathcal{M} = \langle X, Z, Q, \delta, q_0, z_0, \{\bar{q}\}\rangle$ is normalized and birooted. Let $\psi : X^* \to Y^*$ be a monoid homomorphism such that $\psi(X) \subseteq Y$ and let $\bar{\psi} = \psi \circ \psi^{-1}$ ($\bar{\psi}$, the kernel of $\psi$, is a strong relational morphism which is also an equivalence relation; this additional property will be used in what follows). Let $\Gamma$ be the computation 2-graph of $\mathcal{M}$ and let us suppose $\Gamma$ is $\bar{\psi}$-saturated.

Let $\theta : V_\Gamma \to \mathsf{DRB}\langle\langle\ V\ \rangle\rangle$ be the mapping defined by the following: $\forall q \in Q$, $\forall \omega \in Z^*$, such that $q\omega \in V_\Gamma$,

$$\theta(q\omega) = \varphi_0([q\omega\bar{q}]).$$

For every $q\omega \in V_\Gamma$, $S \in \mathsf{DRB}\langle\langle\ V\ \rangle\rangle$ we also define

$$\mathrm{L}(q\omega) = \{u \in X^*, q\omega \xrightarrow{u}_\Gamma \bar{q}, \}; \quad \mathrm{L}(S) = \{u \in X^*, S \odot u = \epsilon\}.$$

LEMMA 3.24. *For every $q\omega \in V_\Gamma$, $\mathrm{L}(q\omega) = \mathrm{L}(\varphi_0([q\omega\bar{q}]))$.*

This lemma follows from the classical result that the language recognized by $\mathcal{M}$ with starting configuration $q\omega$ and final configuration $\bar{q}$ is exactly the language generated by $G_\mathcal{M}$ from the polynomial $[q\omega\bar{q}]$, which, in turn, is equal to the language generated by $G_0$ from the polynomial $\varphi_0([q\omega\bar{q}])$. At last, $G$ and $G_0$ generate the same language from any given polynomial over $V_0$.

LEMMA 3.25. *Let $v, v'$ be vertices of $\Gamma$. Then $v \sim v'$, in the sense of Definition* 2.1 *iff $\theta(v) \sim \theta(v')$, in the sense of Definition* 3.20.

*Proof.* In this proof we denote by $\odot_\Gamma$ the right-action of $X^*$ over $V_\Gamma \cup \{\perp\}$ defined by the following: for every $v, v' \in V_\Gamma, u \in X^*$,

$$v \odot_\Gamma u = v' \text{ if } v \xrightarrow{u}_\Gamma v',$$
$$v \odot_\Gamma u = \perp \text{ if there is no } v'', \text{ such that } v \xrightarrow{u}_\Gamma v'',$$
$$\perp \odot_\Gamma u = \perp.$$

*Step* 1. Let us suppose that $(v, v') \in \mathcal{R}$, where $\mathcal{R}$ is some $\bar{\psi}$-bisimulation over $\Gamma$.

Let $\mathcal{S} = \{(\theta(v) \odot u, \theta(v') \odot u') \mid (u, u') \in \bar{\psi}, (v \odot_\Gamma u, v' \odot_\Gamma u') \in \mathcal{R}\} \cup \{(\emptyset, \emptyset)\}$. Let us show that $\mathcal{S}$ is a $\sigma$-$\bar{\psi}$-bisimulation.

Let us consider some pair of series in $\mathcal{S}$. If the given pair is $(\emptyset, \emptyset)$, points (1), (2) of Definition 3.20 are clearly fulfilled. Otherwise, it has the form $(\theta(v) \odot u, \theta(v') \odot u')$, where $(u, u') \in \bar{\psi}$ and $(v \odot_\Gamma u, v' \odot_\Gamma u') \in \mathcal{R}$.

*Step* 1.1. Let $x \in X$.

*Case* 1.1.1. $\theta(v) \odot ux \neq \emptyset$.

$$\mathrm{L}(\theta(v) \odot ux) \neq \emptyset$$

(because the grammar $G$ is reduced); hence, using Lemma 3.24,

$$\mathrm{L}(v \odot_\Gamma ux) = \mathrm{L}(v) \bullet ux = \mathrm{L}(\theta(v)) \bullet ux \neq \emptyset.$$

It follows that

$$v \odot_\Gamma ux \neq \perp.$$

As $\mathcal{R}$ is a $\bar{\psi}$-simulation, there must exist some $x' \in \bar{\psi}(x)$ such that

$$(v \odot_\Gamma ux, v' \odot_\Gamma u'x') \in \mathcal{R}.$$

Hence

$$(\theta(v) \odot ux, \theta(v') \odot u'x') \in \mathcal{S}.$$

*Case* 1.1.2. $\theta(v) \odot ux = \emptyset$. In this case, by Lemma 3.24 and the fact that $\Gamma$ is birooted, $v \odot_\Gamma ux$ must be equal to $\perp$. As $\Gamma$ is $\bar{\psi}$-saturated, it follows that

$$\forall x' \in \bar{\psi}(x), \quad v \odot_\Gamma ux' = \perp.$$

As $\mathcal{R}^{-1}$ is a $\bar{\psi}^{-1}$-simulation, it must also be true that

$$\forall x' \in \bar{\psi}(x), \quad v' \odot_\Gamma u'x' = \perp.$$

Choosing some particular $x' \in \bar{\psi}(x)$, and again using Lemma 3.24, we obtain

$$\theta(v') \odot u'x' = \emptyset.$$

In both cases, as $v, v'$ are playing symmetric roles, property (1) of Definition 3.20 has been verified. If the starting pair in $\mathcal{S}$ is $(\emptyset, \emptyset)$, property (1) is again verified.

*Step* 1.2. Let us suppose that $\theta(v) \odot u = \epsilon$. This means that

$$\mathrm{L}(\theta(v)) \bullet u = \epsilon,$$

and hence, using Lemma 3.24, that

$$\mathrm{L}(v \odot_\Gamma u) = \epsilon;$$

hence

$$v \odot_\Gamma u = \bar{q}.$$

As $\Gamma$ is birooted, $\bar{q}$ is the only vertex having no outgoing edge (see section 2.1). As $\mathcal{R}$ is a $\bar{\psi}$-bisimulation, $v' \odot_\Gamma u'$, we must also have no outgoing edge; hence

$$v' \odot_\Gamma u' = \bar{q},$$

and by the same arguments, used backwards now,

$$\mathrm{L}(\theta(v')) \bullet u' = \epsilon,$$

which, as the grammar $G$ is proper and reduced, implies

$$\theta(v') \odot u' = \epsilon.$$

As $(v, v')$ are playing symmetric roles, property (2) of Definition 3.20 has been verified.

*Step* 2. Let us suppose that $(\theta(v),\theta(v')) \in \mathcal{S}$, where $\mathcal{S}$ is some $\sigma$-$\bar{\psi}$-bisimulation.

Let $\mathcal{R} = \{(v \odot_\Gamma u, v' \odot_\Gamma u') \mid (u,u') \in \bar{\psi}, (\theta(v) \odot u, \theta(v') \odot u') \in \mathcal{S} - \{(\emptyset,\emptyset)\}\} \cup \{(c,c) \mid c \in V_\Gamma\}$. We show that $\mathcal{R}$ is a $\bar{\psi}$-bisimulation over $\Gamma$.

*Step* 2.1. Using Lemma 3.24, we obtain

$$\theta(v) \odot u \neq \emptyset \Rightarrow v \odot_\Gamma u \neq \perp.$$

Hence

$$\mathrm{dom}(\mathcal{R}) \subseteq V_\Gamma.$$

Conversely, due to the term $\{(c,c) \mid c \in V_\Gamma\}$,

$$\mathrm{dom}(\mathcal{R}) \supseteq V_\Gamma.$$

Finally, point (1) of Definition 2.1 is fulfilled.

*Step* 2.2. Due to the term $\{(c,c) \mid c \in V_\Gamma\}$, point (2) of Definition 2.1 is fulfilled.

*Step* 2.3. Let us consider some pair of configurations in $\mathcal{R}$. It must have the form $(v \odot_\Gamma u, v' \odot_\Gamma u')$, where $(u,u') \in \bar{\psi}$ and $(\theta(v) \odot u, \theta(v') \odot u') \in \mathcal{S} - \{(\emptyset,\emptyset)\}$.

By the same arguments as in Case 1.1.1 above, one can show that, for every $x \in X$, such that

$$v \odot_\Gamma ux \neq \perp,$$

there exists some $x' \in \bar{\psi}(x)$ such that

$$v' \odot_\Gamma u'x' \neq \perp.$$

Hence $\mathcal{R}$ fulfills the three points of Definition 2.1. By the same means, $\mathcal{R}^{-1}$ fulfills them too, so that $\mathcal{R}$ is a $\bar{\psi}$-bisimulation over the graph $\Gamma$. $\square$

**Extension to matrices.** Let $\delta, \lambda \in \mathbb{N} - \{0\}$. We extend the binary relation $\sim$ from vectors in $\mathsf{DRB}_{1,\lambda}\langle\langle\ V\ \rangle\rangle$ to matrices in $\mathsf{DRB}_{\delta,\lambda}\langle\langle\ V\ \rangle\rangle$ as follows: for every $T, T' \in \mathsf{DRB}_{\delta,\lambda}\langle\langle\ V\ \rangle\rangle$,

$$T \sim T' \Leftrightarrow \forall i \in [1,\delta], T_{i,*} \sim T'_{i,*}. \tag{3.7}$$

We call a *w-$\eta$-bisimulation* of order $n \in \mathbb{N} \cup \{\infty\}$ with respect to $(T,T')$ every

$$\mathcal{R} = (\mathcal{R}_i)_{i \in [1,\delta]} \text{ such that } \forall i \in [1,\delta],\ \mathcal{R}_i \in \mathcal{B}_n(T_{i,*}, T'_{i,*}).$$

We denote by $\mathcal{B}_n(T,T')$ the set of $w$-$\eta$-bisimulations of order $n$ w.r.t. $(T,T')$.

Some algebraic properties of this extended relation $\sim$ will be established in Corollary 4.10.

**Operations on $w$-bisimulations.** The following operations on word-$\bar{\psi}$-bisimulations turn out to be useful.

*Right-product.* Let $\delta, \lambda \in \mathbb{N} - \{0\}$, $S, S' \in \mathsf{DRB}_{1,\delta}\langle\langle\ V\ \rangle\rangle$, $T \in \mathsf{DRB}_{\delta,\lambda}\langle\langle\ V\ \rangle\rangle$. For every $n \in \mathbb{N} \cup \{\infty\}$ and $\mathcal{R} \in \mathcal{B}_n(S,S')$ we define

$$\begin{aligned} \langle S|\mathcal{R}\rangle = [&\{(u,u') \in \mathcal{R} \mid \forall v \preceq u, \forall i \in [1,\delta], S \odot v \neq \epsilon_i^\delta\} \\ &\cup \{(u \cdot w, u' \cdot w) \mid (u,u') \in \mathcal{R}, w \in X^*, \exists i \in [1,\delta], S \odot u = \epsilon_i^\delta\}] \\ &\cap X^{\leq n} \times X^{\leq n}. \end{aligned} \tag{3.8}$$

One can check that $\langle S|\mathcal{R}\rangle \in \mathcal{B}_n(S \cdot T, S' \cdot T)$.

*Left-product.* Let $\delta, \lambda \in \mathbb{N} - \{0\}$, $S \in \mathsf{DRB}_{1,\delta}\langle\langle\ V\ \rangle\rangle$, $T, T' \in \mathsf{DRB}_{\delta,\lambda}\langle\langle\ V\ \rangle\rangle$. For every $n \in \mathbb{N} \cup \{\infty\}$ and $\mathcal{R} \in \mathcal{B}_n(T, T')$ we define

$$\begin{aligned}\langle S, \mathcal{R}\rangle = [&\{(u,u) \mid u \in X^*, \forall v \preceq u, \forall i \in [1,\delta], S \odot v \neq \epsilon_i^\delta\}\\ &\cup \{(u\cdot w, u \cdot w') \mid u \in X^*, \exists i \in [1,\delta], S \odot u = \epsilon_i^\delta, (w,w') \in \mathcal{R}_i\}]\\ &\cap X^{\leq n} \times X^{\leq n}.\end{aligned} \tag{3.9}$$

One can check that $\langle S, \mathcal{R}\rangle \in \mathcal{B}_n(S \cdot T, S \cdot T')$.

*Star.* Let $\lambda \in \mathbb{N} - \{0\}$, $S_1 \in \mathsf{DRB}_{1,1}\langle\langle\ V\ \rangle\rangle$, $S_1 \neq \epsilon$, $(S_1, S) \in \mathsf{DRB}_{1,\lambda+1}\langle\langle\ V\ \rangle\rangle$, $T \in \mathsf{DRB}_{1,\lambda}\langle\langle\ V\ \rangle\rangle$. For every $n \in \mathbb{N} \cup \{\infty\}$ and $\mathcal{R} \in \mathcal{B}_n(S_1 \cdot T + S, T)$ we define

$$\mathcal{R}_0 = \mathcal{R}, \tag{3.10}$$

$$\mathcal{S}_0 = \begin{pmatrix} \mathcal{R}_0 \\ \vdots \\ \mathcal{R}_0 \end{pmatrix}, \tag{3.11}$$

$$\forall k \geq 0, \mathcal{R}_{k+1} = \langle (S_1, S), \mathcal{S}_k \rangle \circ \mathcal{R}_0, \tag{3.12}$$

$$\mathcal{S}_k = \begin{pmatrix} \mathcal{R}_k \\ \vdots \\ \mathcal{R}_k \end{pmatrix}, \tag{3.13}$$

and finally

$$\mathcal{R}^{\langle S_1, *\rangle} = \bigcup_{k \geq 0} \mathcal{R}_k \cap X^{\leq k} \times X^{\leq k}. \tag{3.14}$$

One can check that, for every $k \geq 0$,

$$\mathcal{R}_k \in \mathcal{B}_n\left(S_1^{k+1} + \sum_{i=0}^{k} S_1^i \cdot S, T\right), \tag{3.15}$$

$$\mathcal{S}_k \in \mathcal{B}_n\left(\begin{pmatrix} S_1^{k+1} + \sum_{i=0}^{k} S_1^i \cdot S \\ \mathrm{I}_\lambda \end{pmatrix}, \begin{pmatrix} T \\ \mathrm{I}_\lambda \end{pmatrix}\right), \tag{3.16}$$

and finally $\mathcal{R}^{\langle S_1, *\rangle} \in \mathcal{B}_n(S_1^* \cdot S, T)$.

REMARK 3.26. *In fact operations could be more adequately defined on "pointed" w-bisimulations, i.e., on binary relations with sets of "terminal pairs of words" of type $i \in [1, \delta]$ corresponding to the pairs $(u, u')$ such that $S \odot u = \epsilon_i^\delta$, $S' \odot u' = \epsilon_i^\delta$. The two different external operations $\langle S, \mathcal{R}\rangle, \langle S|\mathcal{R}\rangle$ could then be replaced by only one binary operation $\langle \mathcal{R}_1, \mathcal{R}_2\rangle$ over "pointed" w-bisimulations.*

**3.3. Derivations.** For every $u \in X^*$ we define the binary relation $\uparrow(u)$ over $\mathsf{DB}_{1,\lambda}\langle\langle\ V\ \rangle\rangle$ by the following: for every $S, S' \in \mathsf{DB}_{1,\lambda}\langle\langle\ V\ \rangle\rangle$, $S \uparrow (u) S' \Leftrightarrow \exists q \in \mathbb{N}$, $\exists E_1, \ldots, E_k, \ldots, E_q \in V$, $\Phi \in \mathsf{DB}_{q,\lambda}\langle\langle\ V\ \rangle\rangle$ such that

$$S = \sum_{k=1}^{q} E_k \cdot \Phi_k, \qquad S' = \sum_{k=1}^{q} (E_k \odot u) \cdot \Phi_k,$$

and $\forall k \in [1, q]$, $E_1 \smile E_k$, $E_k \odot u \notin \{\emptyset, \epsilon\}$.

It is clear that if $S \uparrow (u)S'$, then $S \odot u = S'$ and that the converse is not true in general. A sequence of deterministic row-vectors $S_0, S_1, \dots, S_n$ is a *derivation* iff there exist $x_1, \dots, x_n \in X$ such that $S_0 \odot x_1 = S_1, \dots, S_{n-1} \odot x_n = S_n$. The *length* of this derivation is $n$. If $u = x_1 \cdot x_2 \cdot \dots \cdot x_n$, we call $S_0, S_1, \dots, S_n$ the derivation *associated* with $(S, u)$. We denote this derivation by $S_0 \xrightarrow{u} S_n$.

A derivation $S_0, S_1, \dots, S_n$ is said to be *stacking* iff it is the derivation associated to a pair $(S, u)$ such that $S = S_0$ and $S_0 \uparrow (u)S_n$. A derivation $S_0, S_1, \dots, S_n$ is said to be a *subderivation* of a derivation $S'_0, S'_1, \dots, S'_m$ iff there exists some $i \in [0, m]$ such that, $\forall j \in [1, n]$, $S_j = S'_{i+j}$.

DEFINITION 3.27. *A vector $S \in \mathsf{DRB}_{1,\lambda}\langle\langle V \rangle\rangle$ is said to be* loop-free *iff for every $v \in V^+$, $S \bullet v \neq S$.*

Let us note that every polynomial is loop-free. The following two lemmas give other examples of loop-free vectors.

LEMMA 3.28. *Let $\alpha \in \mathsf{DB}_{1,n}\langle V \rangle$, $\Phi \in \mathsf{B}_{n,\lambda}\langle\langle V \rangle\rangle$, such that $\infty > \|\alpha \cdot \Phi\| > \|\Phi\|$. Then $\alpha \cdot \Phi$ is loop-free.*

*Proof.* Let $\alpha, \Phi$ fulfill the hypothesis of the lemma and suppose, for the sake of contradiction, that there exists some $v \in V^+$ such that

$$(\alpha \cdot \Phi) \bullet v = \alpha \cdot \Phi.$$

By induction, for every $n \geq 0$,

$$(\alpha \cdot \Phi) \bullet v^n = \alpha \cdot \Phi. \tag{3.17}$$

As $\alpha$ is a polynomial, there exists some $n_0 \geq 0$ such that $|v^{n_0}|$ is greater than the greatest length of a monomial of $\alpha$. Using Lemma 3.10, equality (3.17) for such an integer $n_0$ means that there exists some $k \in [1, n]$, $v''$ suffix of $v^{n_0}$ such that

$$\Phi_k \bullet v'' = \alpha \cdot \Phi. \tag{3.18}$$

Using the hypothesis of the lemma, we conclude that

$$\|\Phi\| \geq \|\Phi_k \bullet v''\| = \|\alpha \cdot \Phi\| > \|\Phi\|,$$

which is contradictory. □

LEMMA 3.29. *Let $S \in \mathsf{DRB}_{1,\lambda}\langle\langle V \rangle\rangle$, $u \in X^*$, such that $\|S \odot u\| > \|S\|$. Then $S \odot u$ is loop-free.*

*Proof.* Let us consider $S, u$ fulfilling the hypothesis of the lemma and let us consider the three possible forms of $S \odot u$ proposed by Lemma 3.14. The forms (1) or (2) are incompatible with the inequality $\|S \odot u\| > \|S\|$. Hence $S \odot u$ has the form (3):

$$u = u_1 \cdot u_2, \quad S \odot u_1 = S \bullet v_1 = \sum_{k=1}^{q} E_k \cdot \Phi_k, \quad S \odot u = \sum_{k=1}^{q} (E_k \odot u_2) \cdot \Phi_k, \quad \text{and}$$

$$\forall k \in [1, q], E_k \smile E_1, E_k \odot u_2 \notin \{\epsilon, \emptyset\}.$$

Hence $S \odot u = \alpha \cdot \Phi$ for some polynomial $\alpha \in \mathsf{DRB}_{1,q}\langle V \rangle$. As for every $k$, $\Phi_k = S \bullet (v_1 E_k)$, we obtain that $\|S\| \geq \|\Phi\|$. Finally

$$\infty > \|S \odot u\| = \|\alpha \cdot \Phi\| > \|S\| \geq \|\Phi\|,$$

and by Lemma 3.28, $S \odot u$ is loop-free. □

Lemma 3.30. *Let $S \in \mathsf{DRB}_{1,\lambda}\langle\langle\ V\ \rangle\rangle$, $w \in X^*$, such that*

(1) *$S$ is loop-free,*

(2) *$\forall u \preceq w$, $\|S \odot u\| \geq \|S\|$.*

*Then the derivation $S \xrightarrow{w} S \odot w$ is stacking.*

*Proof.* $S$ is left-deterministic. If it has type $\emptyset$ or $(\epsilon, j)$, the lemma is trivially true. Otherwise

$$S = \sum_{k=1}^{q} E_k \cdot \Phi_k$$

for some class of letter $[E_1]_\smile = \{E_1, \ldots, E_q\}$ and some matrix $\Phi \in \mathsf{DRB}_{q,\lambda}\langle\langle\ V\ \rangle\rangle$. Suppose that for some prefix $u \preceq w$ and $k \in [1, q]$,

$$E_k \odot u = \epsilon. \tag{3.19}$$

Then $S \odot u = \Phi_k$ so that $\|S \odot u\| \leq \|\Phi\| \leq \|S\|$, which shows that $S = S \odot u$ while $u \neq \epsilon$. This would contradict the hypothesis that $S$ is loop-free; hence (3.19) is impossible.

Let us apply now Lemma 3.15 to the expression $(E \cdot \Phi) \odot w$: case (2) is impossible, and hence

$$(E \cdot \Phi) \odot w = (E \odot w) \cdot \Phi,$$

which is equivalent to

$$S \uparrow (w) S \odot w. \qquad \square$$

Lemma 3.31. *Let $S \in \mathsf{DRB}_{1,\lambda}\langle\langle\ V\ \rangle\rangle$, $w \in X^*$, $k \in \mathbb{N}$, such that*

$$\|S \odot w\| \geq \|S\| + k \cdot K_0 + 1.$$

*Then the derivation $S \xrightarrow{w} S \odot w$ contains some stacking subderivation of length $k$.*

*Sketch of proof.* Let $S = S_0, \ldots, S_i, \ldots, S_n$ be the derivation associated to $(S, w)$. Let $i_0 = \max\{i \in [0, n] \mid \|S_i\| = \min\{\|S_j\| \mid 0 \leq j \leq n\}\}$ and $i_1 = \max\{i \in [i_0+1, n] \mid \|S_i\| = \min\{\|S_j\| \mid i_0 + 1 \leq j \leq n\}\}$. Let $w = w_0 w_1 w'$, where $|w_0| = i_0$, $|w_0 w_1| = i_1$.

As $\|S \odot w_0 w_1\| > \|S \odot w_0\|$, by Lemma 3.29 $S \odot w_0 w_1 = S_{i_1}$ is loop-free. Using Lemma 3.13,

$$\|S_n\| - \|S_{i_1}\| \geq \|S_n\| - \|S_{i_0}\| - (\|S_{i_1}\| - \|S_0\|) \geq (k-1) \cdot K_0 + 1.$$

Using Lemma 3.13 we must have $|w'| \geq k$. Let $w' = w_2 w_3$ with $|w_2| = k$. By definition of $i_1$, $\forall i \in [i_1 + 1, i_1 + k]$, $\|S_i\| \geq \|S_{i_1}\| + 1$.

By Lemma 3.30, the subderivation $S_{i_1}, \ldots, S_{i_1+k}$ (associated to $(S_{i_1}, w_2)$) is stacking. $\square$

Lemma 3.32. *Let $S, S' \in \mathsf{DRB}_{1,\lambda}\langle\langle\ V\ \rangle\rangle$, $w \in X^*$, $k, d, d' \in \mathbb{N}$, such that $S$ is d-marked and*

(1) *the derivation $S \xrightarrow{w} S'$ contains no stacking subderivation of length $k$,*

(2) *$|w| \geq d \cdot k$.*

*Then $S'$ is unmarked.*

*Proof.* By the hypothesis,

$$S = \sum_{k=1}^{q} \alpha_k \cdot \Phi_k$$

for some $\alpha \in \mathsf{DRB}_{1,q}\langle\ V\ \rangle$, $\Phi \in \mathsf{DRB}_{q,\lambda}\langle\langle\ V\ \rangle\rangle$, $\|\alpha\| \leq d$, $\Phi$ unmarked.

Let $S \xrightarrow{w} S' = (S_0, \ldots, S_n)$. By induction on $\ell$, using hypothesis (1) and Lemma 3.30 (on polynomials, which are particular cases of loop-free series) one can show that for every $\ell \in [0, d]$, there exists some prefix $w_\ell$ of $w$, with length $|w_\ell| \leq k \cdot \ell$, such that either

$$S \odot w_\ell = \sum_{k=1}^{q} (\alpha_k \odot w_\ell) \cdot \Phi_k \text{ with } \|\alpha_\odot w_\ell\| < \|\alpha\| - \ell \tag{3.20}$$

or there exists an integer $k \in [1, q]$ such that

$$S \odot w_\ell = \Phi_k. \tag{3.21}$$

Let us apply this property to $\ell = d$: inequality (3.20) is not possible for this value of $\ell$ because, by hypothesis (2) of the lemma, $\|\alpha\| - \ell \leq 0$. Hence (3.21) is true and, as $\Phi$ is unmarked, $\Phi_k$ is unmarked, so that $S \odot w$ is unmarked. $\square$

## 4. Deduction systems.

### 4.1. General formal systems.

We follow here the general philosophy of [16, 9]. Let us call a *formal system* any triple $\mathcal{D} = \langle \mathcal{A}, H, \models\!\!- \rangle$ where $\mathcal{A}$ is a denumerable set called the *set of assertions*, $H$, the *cost function*, is a mapping $\mathcal{A} \to \mathbb{N} \cup \{\infty\}$, and $\models\!\!-$, the *deduction relation*, is a subset of $\mathcal{P}_f(\mathcal{A}) \times \mathcal{A}$; $\mathcal{A}$ is given with a fixed bijection with $\mathbb{N}$ (an "encoding" or "Gödel numbering") so that the notions of recursive subset, recursively enumerable subset, recursive function, over $\mathcal{A}, \mathcal{P}_f(\mathcal{A})$, are defined, up to this fixed bijection; we assume that $\mathcal{D}$ satisfies the following axioms:

(A1) $\forall (P, A) \in \models\!\!-$, $(\min\{H(p), p \in P\} < H(A))$ or $(H(A) = \infty)$.

(We let $\min(\emptyset) = \infty$.) We call $\mathcal{D}$ a *deduction system* iff $\mathcal{D}$ is a formal system satisfying the additional axiom:

(A2) $\models\!\!-$ is recursively enumerable.

In what follows we use the notation $P \models\!\!- A$ for $(P, A) \in \models\!\!-$. We call a *proof* in the system $\mathcal{D}$, *relative to the set of hypotheses* $\mathcal{H} \subseteq \mathcal{A}$, any subset $P \subseteq \mathcal{A}$ fulfilling

$$\forall p \in P,\ (\exists Q \subseteq P, Q \models\!\!- p) \text{ or } (p \in \mathcal{H}).$$

We call $P$ a *proof* iff

$$\forall p \in P, (\exists Q \subseteq P, Q \models\!\!- p)$$

(i.e., iff $P$ is a proof relative to $\emptyset$).

Let us define the total map $\chi : \mathcal{A} \to \{0, 1\}$ and the partial map $\overline{\chi} : \mathcal{A} \to \{0, 1\}$ by

$$\chi(A) = 1 \text{ if } H(A) = \infty, \quad \chi(A) = 0 \text{ if } H(A) < \infty,$$

$$\overline{\chi}(A) = 1 \text{ if } H(A) = \infty, \quad \overline{\chi} \text{ is undefined if } H(A) < \infty.$$

($\chi$ is the "truth-value function"; $\overline{\chi}$ is the "1-value function.")

LEMMA 4.1. *Let $P$ be a proof relative to $\mathcal{H} \subseteq H^{-1}(\infty)$ and $A \in P$. Then $\chi(A) = 1$.*

In other words, if an assertion is provable from true hypotheses, then it is true.

*Proof.* Let $P$ be a proof. We prove by induction on $n$ that

$$\mathcal{P}(n) : \forall p \in P, H(p) \geq n.$$

It is clear that, $\forall p \in P$, $H(p) \geq 0$. Suppose that $\mathcal{P}(n)$ is true. Let $p \in P - \mathcal{H} : \exists Q \subseteq P$, $Q \vdash\!\!- p$. By the induction hypothesis, $\forall q \in Q$, $H(q) \geq n$ and by (A1), $H(p) \geq n+1$. It follows that, $\forall p \in P - \mathcal{H}$, $H(p) = \infty$. But by the hypothesis, $\forall p \in \mathcal{H}$, $H(p) = \infty$. ☐

A formal system $\mathcal{D}$ will be *complete* iff, conversely, $\forall A \in \mathcal{A}$, $\chi(A) = 1 \Longrightarrow$ there exists some *finite* proof $P$ such that $A \in P$. (In other words, $\mathcal{D}$ is complete iff every true assertion is "finitely" provable.)

LEMMA 4.2. *If $\mathcal{D}$ is a complete deduction system, $\overline{\chi}$ is a recursive partial map.*

*Proof.* Let $i \mapsto P_i$ be some recursive function whose domain is $\mathbb{N}$ and whose image is $\mathcal{P}_f(\mathcal{A})$. Let $h : (\mathcal{P}_f(\mathcal{A}) \times \mathcal{A} \times \mathbb{N}) \to \{0,1\}$ be a total recursive function such that

$$P \vdash\!\!- A \text{ iff } \exists n \in \mathbb{N},\ h(P, A, n) = 1$$

(such an $h$ exists, because the recursively enumerable sets are the projections of the recursive sets; see [30]).

The following (informal) semialgorithm computes $\overline{\chi}$ on the assertion A:

```
1. $i$ := 0 ; $n$ := 0 ; $s$ := $i+n$;
2. $P$ := $P_i$;
3. $b$ := $\min_{p \in P}\{\max_{Q \subseteq P}\{h(Q,p,n)\}\}$;
4. $c$ := $(A \in P)$;
5. if $(b \wedge c)$ then $(\overline{\chi}(A) = 1$ ; stop);
6. if $i = 0$ then $(i := s+1$ ; $n := 0$; $s := i+n)$
   else $(i := i-1$ ; $n := n+1)$;
7. goto 2;
```
☐

In order to define deduction relations from more elementary ones, we set the following definitions.

Let $\vdash\!\!- \subseteq \mathcal{P}_f(\mathcal{A}) \times \mathcal{A}$. For every $P, Q \in \mathcal{P}_f(\mathcal{A})$ we set

- $P \overset{[0]}{\vdash\!\!-} Q$ iff $P \supseteq Q$,
- $P \overset{[1]}{\vdash\!\!-} Q$ iff $\forall q \in Q$, $\exists R \subseteq P$, $R \vdash\!\!- q$,
- $P \overset{\langle 0 \rangle}{\vdash\!\!-} Q$ iff $P \overset{[0]}{\vdash\!\!-} Q$,
- $P \overset{\langle 1 \rangle}{\vdash\!\!-} Q$ iff $\forall q \in Q$, $(\exists R \subseteq P,\ R \vdash\!\!- q)$ or $(q \in P)$,
- $P \overset{\langle n+1 \rangle}{\vdash\!\!-} Q$ iff $\exists R \in \mathcal{P}_f(A)$, $P \overset{\langle 1 \rangle}{\vdash\!\!-} R$ and $R \overset{\langle n \rangle}{\vdash\!\!-} Q$ (for every $n \geq 0$),
- $\overset{\langle * \rangle}{\vdash\!\!-} = \bigcup_{n \geq 0} \overset{\langle n \rangle}{\vdash\!\!-}$.

Given $\vdash\!\!-_1, \vdash\!\!-_2 \subseteq \mathcal{P}_f(\mathcal{A}) \times \mathcal{P}_f(\mathcal{A})$, for every $P, Q \in \mathcal{P}_f(\mathcal{A})$ we set

$$P(\vdash\!\!-_1 \circ \vdash\!\!-_2)Q \text{ iff } \exists R \subseteq \mathcal{A},\ (P \vdash\!\!-_1 R) \wedge (R \vdash\!\!-_2 Q).$$

**4.2. Strategies.** We define here a notion of strategy which consists of a map allowing us to build proofs within a formal system $\mathcal{D}$. This notion will be an essential tool for proving that a formal system is *complete.* Let $\mathcal{D} = \langle \mathcal{A}, H, \vdash\!\!- \rangle$ be a formal system. We call a *strategy* for $\mathcal{D}$ any map $\mathcal{S} : \mathcal{A}^+ \to \mathcal{P}(\mathcal{A}^*)$ such that

(S1) if $B_1 \cdots B_m \in \mathcal{S}(A_1 A_2 \cdots A_n)$, then $\exists Q \subseteq \{A_i \mid 1 \leq i \leq n-1\}$ such that

$$\{B_j \mid 1 \leq j \leq m\} \cup Q \vdash\!\!- A_n;$$

(S2) if $B_1 \cdots B_m \in \mathcal{S}(A_1 A_2 \cdots A_n)$, then

$$\min\{H(A_i) \mid 1 \leq i \leq n\} = \infty \Longrightarrow \min\{H(B_j) \mid 1 \leq j \leq m\} = \infty.$$

REMARK 4.3. *It may happen that $\epsilon \in \mathcal{S}(A_1 A_2 \cdots A_n)$ (and, correspondingly, that $m = 0$ in the above conditions): it just means that $\{A_1, \ldots, A_{n-1}\} \mid\!\!-\!\!- A_n$. It may also happen that $\mathcal{S}(A_1 A_2 \cdots A_n) = \emptyset$: this means, intuitively, that $\mathcal{S}$ "does not know" how to extend a proof (with hypothesis), with the only information being that the given proof contains the assertions $A_1, A_2, \ldots, A_n$.*

REMARK 4.4. *Axiom* (A1) *on systems is similar to the "monotonicity" condition of* [16] *or axiom* (2.4.2′) *of* [9]. *Axiom* (S2) *on strategies is similar to the "validity" condition of* [16] *or property* (2.4.1′) *of* [9].

Given a strategy $\mathcal{S}$, we define $\mathcal{T}(\mathcal{S}, A)$, the set of proof-trees associated to the strategy $\mathcal{S}$ and the assertion $A$, as the set of all trees $t$ fulfilling the following properties:

$$\varepsilon \in \mathrm{dom}(t), \quad t(\varepsilon) = A, \tag{4.1}$$

and, for every path $x_0 x_1, \ldots, x_{n-1}$ in $t$, with labels $t(x_i) = A_{i+1}$ (for $0 \le i \le n-1$), if $x_{n-1}$ has $m$ sons $x_{n-1} \cdot 1, \ldots, x_{n-1} \cdot m \in \mathrm{dom}(t)$ with labels $t(x_{n-1} \cdot j) = B_j$ (for $1 \le j \le m$), then

$$(B_1 \cdots B_m) \in \mathcal{S}(A_1 \cdots A_n) \quad \text{or} \quad m = 0. \tag{4.2}$$

The proof-tree $t$ is said to be *closed* iff it fulfills the following additional condition: for every path $x_0 x_1, \ldots, x_{n-1}$ in $t$, with labels $t(x_i) = A_{i+1}$ (for $0 \le i \le n-1$), if $x_{n-1}$ has $m$ sons $x_{n-1} \cdot 1, \ldots, x_{n-1} \cdot m \in \mathrm{dom}(t)$ with labels $t(x_{n-1} \cdot j) = B_j$ (for $1 \le j \le m$), then

$$m = 0 \Rightarrow ((\exists i \in [1, n-1],\, A_i = A_n) \text{ or } (\varepsilon \in \mathcal{S}(A_1 \cdots A_n))). \tag{4.3}$$

A node $x \in \mathrm{dom}(t)$ is said to be *closed* iff it is an internal node or it is a leaf fulfilling property (4.3) above.

The proof-tree $t$ is said to be *repetition-free* iff, for every $x, x' \in \mathrm{dom}(t)$,

$$[x \preceq x' \text{ and } t(x) = t(x')] \quad \Rightarrow \quad x = x' \text{ or } x' \text{ is a leaf.}$$

For every tree $t$ let us define

$$\mathcal{L}(t) = \{t(x) \mid \forall y \in \mathrm{dom}(t), x \preceq y \Rightarrow x = y\}, \quad \mathcal{I}(t) = \{t(x) \mid \exists y \in \mathrm{dom}(t), x \prec y\}.$$

(Here $\mathcal{L}$ stands for "leaves" and $\mathcal{I}$ stands for "internal nodes.")

LEMMA 4.5. *If $\mathcal{S}$ is a strategy for the deduction-system $\mathcal{D}$, then, for every true assertion $A$ and every $t \in \mathcal{T}(\mathcal{S}, A)$,*

(1) *the set of labels of $t$ is a $\mathcal{D}$-proof, relative to the set $\mathcal{L}(t) - \mathcal{I}(t)$;*

(2) *every label of a leaf is true.*

*Proof.* Let us suppose that $H(A) = \infty$. Let $t \in \mathcal{T}(\mathcal{S}, A)$, $P = \mathrm{im}(t)$ (the set of labels of $t$), $\mathcal{H} = \mathcal{L}(t) - \mathcal{I}(t)$.

Using (S2), one can prove by induction on the depth of $x \in \mathrm{dom}(t)$ that $H(t(x)) = \infty$. Point (2) is then proved. Let $x$ be an internal node of $t$, with sons $x \cdot 1, x \cdot 2, \ldots, x \cdot m$ ($m \ge 1$), and with ancestors $y_1, y_2, \ldots, y_{n-1}, y_n = x$ ($n \ge 1$), such that

$$t(y_1) \cdots t(y_n) = A_1 \cdots A_n, \quad t(x_1) \cdots t(x_m) = B_1 \cdots B_m.$$

By definition of $\mathcal{T}(\mathcal{S}, A)$,

$$B_1 \cdots B_m \in \mathcal{S}(A_1 \cdots A_n),$$

and by condition (S1),

$$\exists Q \subseteq \{A_i \mid i \leq n-1\}, \text{ such that } \{B_j \mid 1 \leq j \leq m\} \cup Q \mathrel{|\!\!-\!\!-} A_n.$$

It follows that for every $p \notin \mathcal{H}$, $\exists R \subseteq P$, $R \mathrel{|\!\!-\!\!-} p$; hence

$$\forall p \in P, \quad (\exists R \subseteq P,\ R \mathrel{|\!\!-\!\!-} p) \text{ or } p \in \mathcal{H}.$$

Point (1) is proved. ∎

For every $\mathcal{D}$-strategy $\mathcal{S}$, we use the notation

$$\mathcal{T}(\mathcal{S}) = \bigcup_{A \in H^{-1}(\infty)} \mathcal{T}(\mathcal{S}, A).$$

We call a *global strategy* w.r.t. $\mathcal{S}$ any total map $\hat{\mathcal{S}} : \mathcal{T}(\mathcal{S}) \to \mathcal{T}(\mathcal{S})$ such that

$$\forall t \in \mathcal{T}(\mathcal{S}), \quad t \preceq \hat{\mathcal{S}}(t). \tag{4.4}$$

$\hat{\mathcal{S}}$ is a *terminating* global strategy iff

$$\forall A_0 \in H^{-1}(\infty),\ \exists n_0 \in \mathbb{N}, \quad \hat{\mathcal{S}}^{n_0}(A_0) = \hat{\mathcal{S}}^{n_0+1}(A_0), \tag{4.5}$$

and $\hat{\mathcal{S}}$ is a *closed* global strategy iff

$$\forall A_0 \in H^{-1}(\infty), \forall n \in \mathbb{N}, \quad \hat{\mathcal{S}}^{n}(A_0) \text{ is closed} \iff \hat{\mathcal{S}}^{n}(A_0) = \hat{\mathcal{S}}^{n+1}(A_0) \tag{4.6}$$

(where the assertion $A_0$ is identified with the tree reduced to one node whose label is $A_0$).

LEMMA 4.6. *Let $\mathcal{D}$ be a formal system, $\mathcal{S}$ a strategy for $\mathcal{D}$, and $\hat{\mathcal{S}}$ a global strategy w.r.t. $\mathcal{S}$. If $\hat{\mathcal{S}}$ is terminating and $\hat{\mathcal{S}}$ is closed, then $\mathcal{D}$ is complete.*

*Proof.* Let $A_0 \in \mathcal{A}$. Under the hypothesis of the lemma, $\exists n_0 \in \mathbb{N}$ such that (4.5) and (4.6) are both true. Hence $t_\infty = \hat{\mathcal{S}}^{n_0}(A_0)$ is a closed proof-tree for $\mathcal{S}$. By Lemma 4.5 $\mathrm{im}(t_\infty)$ is a $\mathcal{D}$-proof relative to the set $\mathcal{L}(t_\infty) - \mathcal{I}(t_\infty)$. Let $x$ be a leaf such that $t_\infty(x) \in \mathcal{L}(t_\infty) - \mathcal{I}(t_\infty)$. Let $A_0, A_1, \ldots, A_n = t_\infty(x)$ be the word labeling the path from the root to $x$. As $x$ is closed and $t_\infty(x) \in \mathcal{L}(t_\infty) - \mathcal{I}(t_\infty)$ by (4.3), $\varepsilon \in \mathcal{S}(A_1 \cdots A_n)$; hence $\{A_1, \ldots, A_{n-1}\} \mathrel{|\!\!-\!\!-} t_\infty(x)$. It follows that $\mathrm{im}(t_\infty)$ is a $\mathcal{D}$-proof. ∎

**4.3. System $\mathcal{B}_0$.** Let us define here a particular formal system $\mathcal{B}_0$ "tailored for the $\sigma$-$\bar{\psi}$-bisimulation problem for deterministic series."

Let us fix two finite alphabets $X, Y$, a surjection $\psi : X \to Y$ (which induces a surjection $X^* \to Y^*$ denoted by the same symbol $\psi$), and its kernel $\bar{\psi} = \mathrm{Ker}\,\psi \subseteq X^* \times X^*$ (see section 3.2). We also fix a dpda $\mathcal{M}$ over the terminal alphabet $X$ and consider the variable alphabet $V$ associated to $\mathcal{M}$ (see section 3.1) and the sets $\mathsf{DRB}_{\delta,\lambda}\langle\langle V \rangle\rangle$ (the sets of deterministic rational boolean matrices over $V^*$, with $\delta$ rows and $\lambda$ columns). The set of assertions is defined by

$$\mathcal{A} = \bigcup_{\lambda \geq 1} \mathbb{N} \times \mathsf{DRB}_{1,\lambda}\langle\langle V \rangle\rangle \times \mathsf{DRB}_{1,\lambda}\langle\langle V \rangle\rangle;$$

i.e., an assertion is here a *weighted equation* over $\mathsf{DRB}_{1,\lambda}\langle\langle V \rangle\rangle$ for some integer $\lambda$.

For every $n \geq 0$ we define

$$\bar{\mathcal{B}}_n = \{\mathcal{R} \subseteq \bar{\psi} \mid \mathcal{R} \text{ fulfills conditions } (1'), (2'), \text{ and } (4) \text{ of Definition 3.21}\}. \tag{4.7}$$

We call the elements of $\bar{\mathcal{B}}_n$ the *admissible* relations of order $n$ over $X^* \times X^*$. For every pair $(S, S') \in \mathsf{DRB}_{1,\lambda}\langle\langle\, V\,\rangle\rangle \times \mathsf{DRB}_{1,\lambda}\langle\langle\, V\,\rangle\rangle$, and $n \in \mathbb{N} \cup \{\infty\}$, we define

$$\mathcal{B}_n(S, S') = \{\mathcal{R} \subseteq \bar{\psi} \mid \mathcal{R} \text{ is a } w\text{-}\bar{\psi}\text{-bisimulation of order } n \text{ w.r.t. } (S, S')\}. \tag{4.8}$$

The "cost-function" $H : \mathcal{A} \to \mathbb{N} \cup \{\infty\}$ is defined by

$$H(n, S, S') = n + 2 \cdot \mathrm{Div}(S, S'),$$

where $\mathrm{Div}(S, S')$ is the *divergence* between $S$ and $S'$ (Definition 3.23). We recall that it is defined by

$$\mathrm{Div}(S, S') = \inf\{n \in \mathbb{N} \mid \mathcal{B}_n(S, S') = \emptyset\}.$$

(We recall $\inf(\emptyset) = \infty$.)

Let us note that, by Lemma 3.22,

$$\chi(n, S, S') = 1 \Longleftrightarrow S \sim S'.$$

We define a binary relation $|\!\!\vdash\!\!-\; \subseteq \mathcal{P}_f(\mathcal{A}) \times \mathcal{A}$, the *elementary deduction relation*, as the set of all the pairs having one of the following forms:

(R0)

$$\{(p, S, T)\} \;|\!\!\vdash\!\!-\; (p + 1, S, T)$$

for $p \in \mathbb{N}$, $\lambda \in \mathbb{N} - \{0\}$, $S, T \in \mathsf{DRB}_{1,\lambda}\langle\langle\, V\,\rangle\rangle$;

(R1)

$$\{(p, S, T)\} \;|\!\!\vdash\!\!-\; (p, T, S)$$

for $p \in \mathbb{N}$, $\lambda \in \mathbb{N} - \{0\}$, $S, T \in \mathsf{DRB}_{1,\lambda}\langle\langle\, V\,\rangle\rangle$;

(R2)

$$\{(p, S, S'), (p, S', S'')\} \;|\!\!\vdash\!\!-\; (p, S, S'')$$

for $p \in \mathbb{N}$, $\lambda \in \mathbb{N} - \{0\}$, $S, T \in \mathsf{DRB}_{1,\lambda}\langle\langle\, V\,\rangle\rangle$;

(R3)

$$\emptyset \;|\!\!\vdash\!\!-\; (0, S, S)$$

for $S \in \mathsf{DRB}_{1,\lambda}\langle\langle\, V\,\rangle\rangle$;

(R′3)

$$\emptyset \;|\!\!\vdash\!\!-\; (0, S, \rho_e(S))$$

for $S \in \mathsf{DRB}_{1,1}\langle\langle\, V\,\rangle\rangle$;

(R4)

$$\{(p + 1, S \odot x, T \odot x') \mid (x, x') \in \mathcal{R}_1\} \;|\!\!\vdash\!\!-\; (p, S, T)$$

for $p \in \mathbb{N}$, $\lambda \in \mathbb{N} - \{0\}$, $S, T \in \mathsf{DRB}_{1,\lambda}\langle\langle\, V\,\rangle\rangle$, $(S \neq \epsilon \wedge T \neq \epsilon)$, and $\mathcal{R}_1 \in \bar{\mathcal{B}}_1$;

(R5)

$$\{(p, S, S')\} \;|\!\!\vdash\!\!-\; (p + 2, S \odot x, S' \odot x')$$

for $p \in \mathbb{N}$, $\lambda \in \mathbb{N} - \{0\}$, $S, T \in \mathsf{DRB}_{1,\lambda}\langle\langle\, V\,\rangle\rangle$, $(x, x') \in \bar{\psi}$, $S \sim S' \wedge S \odot x \sim S' \odot x'$;

(R6)

$$\{(p, S_1 \cdot T + S, T)\} \,||\!\!-\!\!- (p, S_1^* \cdot S, T)$$

for $p \in \mathbb{N}$, $\lambda \in \mathbb{N}-\{0\}$, $S_1 \in \mathsf{DRB}_{1,1}\langle\langle\, V\,\rangle\rangle$, $S_1 \neq \epsilon$, $(S_1, S) \in \mathsf{DRB}_{1,\lambda+1}\langle\langle\, V\,\rangle\rangle$, $T \in \mathsf{DRB}_{1,\lambda}\langle\langle\, V\,\rangle\rangle$;

(R7)

$$\{(p, S, S')\} \,||\!\!-\!\!- (p, S \cdot T, S' \cdot T)$$

for $p \in \mathbb{N}$, $\delta, \lambda \in \mathbb{N} - \{0\}$, $S, S' \in \mathsf{DRB}_{1,\delta}\langle\langle\, V\,\rangle\rangle$, $T \in \mathsf{DRB}_{\delta,\lambda}\langle\langle\, V\,\rangle\rangle$;

(R8)

$$\{(p, T_{i,*}, T'_{i,*}) \mid 1 \leq i \leq \delta\} \,||\!\!-\!\!- (p, S \cdot T, S \cdot T')$$

for $p \in \mathbb{N}$, $\delta, \lambda \in \mathbb{N} - \{0\}$, $S \in \mathsf{DRB}_{1,\delta}\langle\langle\, V\,\rangle\rangle$, $T, T' \in \mathsf{DRB}_{\delta,\lambda}\langle\langle\, V\,\rangle\rangle$.

REMARK 4.7. *We do not claim that this formal system is* recursively enumerable*: due to rule* (R5)*, establishing this property is as difficult as solving the general bisimulation problem for equational graphs of finite out-degree. This difficulty will be overcome in section* 10 *by an* elimination *lemma.*

LEMMA 4.8. *Let $P \in \mathcal{P}_f(\mathcal{A})$, $A \in \mathcal{A}$, such that $P \,||\!\!-\!\!- A$. Then* $\min\{H(p) \mid p \in P\} \leq H(A)$.

Let us introduce the following notation: for every $n \in \mathbb{N} \cup \{\infty\}$, $\lambda \in \mathbb{N} - \{0\}$, $S, S' \in \mathsf{DRB}_{1,\lambda}\langle\langle\, V\,\rangle\rangle$,

$$S \sim_n S' \Leftrightarrow \mathcal{B}_n(S, S') \neq \emptyset.$$

*Proof.* Let us check this property for every type of rule.

(R0) $p + 2 \cdot \mathrm{Div}(S, T) \leq p + 1 + 2 \cdot \mathrm{Div}(S, T)$.

(R1) $p + 2 \cdot \mathrm{Div}(S, T) = p + 2 \cdot \mathrm{Div}(T, S)$.

(R2) As the weight $p$ is the same in all the considered equations, we are reduced to proving that $\forall n \in \mathbb{N}$, $S \sim_n S' \wedge S' \sim_n S'' \Longrightarrow S \sim_n S''$. This is true because, if $\mathcal{R} \in \mathcal{B}_n(S, S')$ and $\mathcal{R}' \in \mathcal{B}_n(S', S'')$, then $\mathcal{R} \circ \mathcal{R}' \in \mathcal{B}_n(S, S'')$.

(R3) Let us note that $\mathrm{Id}_{X^*} \subseteq \bar{\psi}$. It follows that $\infty = \mathrm{Div}(S, S)$.

(R′3) The definition of $G$ from $G_0$ is such that $S \equiv \rho_e(S)$; hence $S \sim \rho_e(S)$ and $\infty = \mathrm{Div}(S, \rho_e(S))$.

(R4) Let $n \in \mathbb{N}$ such that

$$\forall (x, x') \in \mathcal{R}_1, \quad n \leq \mathrm{Div}(S \odot x, S' \odot x').$$

Let us choose, for every $(x, x') \in \mathcal{R}_1$, some $\mathcal{R}_{x,x'} \in \mathcal{B}_n(S \odot x, S' \odot x')$. Let us then define

$$\mathcal{R} = \bigcup_{(x,x') \in \mathcal{R}_1} (x, x') \cdot \mathcal{R}_{x,x'}.$$

$\mathcal{R}$ belongs to $\mathcal{B}_{n+1}(S, S')$. It follows that

$$\min\{\mathrm{Div}(S \odot x, S' \odot x') \mid (x, x') \in \mathcal{R}_1\} + 1 \leq \mathrm{Div}(S, S'),$$

and hence that

$$\min\{H(p + 1, S \odot x, S' \odot x') \mid (x, x') \in \mathcal{R}_1\} \leq H(p, S, S') - 1.$$

(R5) By the hypothesis, $H(p+2, S \odot x, S' \odot x') = \infty$.

(R6) Let $n \in \mathbb{N}$ such that

$$n \leq \mathrm{Div}(S_1 \cdot T + S, T).$$

Let $\mathcal{R} \in \mathcal{B}_n(S_1 \cdot T + S, T)$. Let $\mathcal{R}' = \mathcal{R}^{\langle S_1, * \rangle}$ (see definition (3.14) in section 3.2). Since we have

$$\mathcal{R}' \in \mathcal{B}_n(S_1^* \cdot S, T),$$

we get the inequality $\mathrm{Div}(S_1 \cdot T + S, T) \leq \mathrm{Div}(S_1^* \cdot S, T)$.

(R7) Let $n \leq \mathrm{Div}(S, S')$ and $\mathcal{R} \in \mathcal{B}_n(S, S')$. Let us consider $\mathcal{R}' = \langle S | \mathcal{R} \rangle$ (see definition (3.8) in section 3.2). Since we have $\mathcal{R}' \in \mathcal{B}_n(S \cdot T, S' \cdot T)$, the required inequality is proved.

(R8) Let $n \leq \min\{\mathrm{Div}(T_{i,*}, T'_{i,*}) \mid 1 \leq i \leq \delta\}$ and, for every $i \in [1, \delta]$, let $\mathcal{R}_i \in \mathcal{B}_n(T_{i,*}, T'_{i,*})$. Let us consider $\mathcal{R}' = \langle S, \mathcal{R} \rangle$ (see definition (3.9) in section 3.2). Since we know that

$$\mathcal{R}' \in \mathcal{B}_n(S \cdot T, S \cdot T'),$$

the required inequality is proved. □

Let us define $\vdash\!\!-$ as follows: for every $P \in \mathcal{P}_f(\mathcal{A})$, $A \in \mathcal{A}$,

$$P \vdash\!\!- A \iff P \overset{\langle * \rangle}{||\!\!\vdash\!\!-} \circ \overset{[1]}{||\!\!\vdash\!\!-}_{0,3,4} \circ \overset{\langle * \rangle}{||\!\!\vdash\!\!-} \{A\},$$

where $||\!\!\vdash\!\!-_{0,3,4}$ is the relation defined by $R_0, R_3, R'_3, R_4$ only. We let

$$\mathcal{B}_0 = \langle \mathcal{A}, H, \vdash\!\!- \rangle.$$

LEMMA 4.9. *$\mathcal{B}_0$ is a formal system.*

*Proof.* Using Lemma 4.8, one can show by induction on $n$ that

$$P \overset{\langle n \rangle}{||\!\!\vdash\!\!-} Q \Longrightarrow \forall q \in Q, \quad \min\{H(A) \mid A \in P\} \leq H(q).$$

The proof of Lemma 4.8 also reveals that

$$P \,||\!\!\vdash\!\!-_{\{0,3,4\}} q \Longrightarrow (\min\{H(p) \mid p \in P\} < H(q)) \quad \text{or} \quad H(q) = \infty.$$

It follows that, for every $m, n \geq 0$,

$$P \overset{\langle n \rangle}{||\!\!\vdash\!\!-} Q \overset{[1]}{||\!\!\vdash\!\!-}_{0,3,4} R \overset{\langle m \rangle}{||\!\!\vdash\!\!-} q \Longrightarrow (\min\{H(p) \mid p \in P\} < H(q)) \quad \text{or} \quad H(q) = \infty.$$

Hence axiom (A1) is fulfilled. □

Let us note the following algebraic corollaries of Lemma 4.8.

COROLLARY 4.10.

(C1) $\forall \lambda \in \mathbb{N} - \{0\}$, $S_1 \in \mathsf{DRB}_{1,1}\langle\langle\, V \,\rangle\rangle$, $S_1 \not\equiv \epsilon$, $(S_1, S) \in \mathsf{DRB}_{1,\lambda+1}\langle\langle\, V \,\rangle\rangle$, $T \in \mathsf{DRB}_{1,\lambda}\langle\langle\, V \,\rangle\rangle$,

$$S_1 \cdot T + S \sim T \Longrightarrow S_1^* \cdot S \sim T.$$

(C2) $\forall \delta, \lambda \in \mathbb{N} - \{0\}$, $S, S' \in \mathsf{DRB}_{1,\delta}\langle\langle\, V \,\rangle\rangle$, $T \in \mathsf{DRB}_{\delta,\lambda}\langle\langle\, V \,\rangle\rangle$,

$$S \sim S' \Longrightarrow S \cdot T \sim S' \cdot T.$$

(C3) $\forall \lambda \in \mathbb{N} - \{0\}$, $S, S' \in \mathsf{DRB}_{1,1}\langle\langle\ V\ \rangle\rangle$, $T \in \mathsf{DRB}_{1,\lambda}\langle\langle\ V\ \rangle\rangle$,

$$[S \cdot T \sim S' \cdot T \text{ and } T \neq \emptyset^{\lambda}] \Longrightarrow S \sim S'.$$

(C4) $\forall \delta, \lambda \in \mathbb{N} - \{0\}$, $S \in \mathsf{DRB}_{1,\delta}\langle\langle\ V\ \rangle\rangle$, $T, T' \in \mathsf{DRB}_{\delta,\lambda}\langle\langle\ V\ \rangle\rangle$,

$$T \sim T' \Longrightarrow S \cdot T \sim S \cdot T'.$$

*Proof.* Statement (C$i$) (for $i \in \{1, 2, 4\}$) is a direct corollary of the fact that the value of $H$ on the left-hand side of some rule (R$j$) is smaller than or equal to the value of $H$ on the right-hand side of rule (R$j$): (C1) is justified by (R6), (C2) by (R7), (C4) by (R8).

Let us prove (C3): Suppose that $\lambda \in \mathbb{N} - \{0\}$, $S, S' \in \mathsf{DRB}_{1,1}\langle\langle\ V\ \rangle\rangle$, $T \in \mathsf{DRB}_{1,\lambda}\langle\langle\ V\ \rangle\rangle$, and

$$S \cdot T \sim S' \cdot T \quad \text{and} \quad S \not\sim S'. \tag{4.9}$$

Let $\mathcal{R} \in \mathcal{B}_{\infty}(S \cdot T, S' \cdot T)$ and let

$$(u, u') = \min\{(v, v') \in \mathcal{R} \mid (S \odot v = \epsilon) \Leftrightarrow (S' \odot v' \neq \epsilon)\}.$$

From the hypothesis that $\mathcal{R} \in \mathcal{B}_{\infty}(S \cdot T, S' \cdot T)$, we get that

$$\forall (v, v') \in \mathcal{R}, \quad (S \cdot T) \odot v \sim (S' \cdot T) \odot v',$$

and by the choice of $(u, u')$ we obtain that

$$T \sim (S' \odot u') \cdot T \quad \text{or} \quad (S \odot u) \cdot T \sim T,$$

which, by (C1), implies

$$T \sim (S' \odot u')^{*} \cdot \emptyset^{\lambda} \quad \text{or} \quad (S \odot u)^{*} \cdot \emptyset^{\lambda} \sim T,$$

i.e., $T \sim \emptyset^{\lambda}$, which implies (because $G$ is a reduced grammar) that

$$T = \emptyset^{\lambda}. \tag{4.10}$$

We have proved that (4.9) implies (4.10), and hence (C3). □

**4.4. Congruence closure.** Let us consider the subset $\mathcal{C}$ of the rules of $\mathcal{B}_0$, consisting of all the instances of the metarules (R0), (R1), (R2), (R3), (R′3), (R6), (R7), (R8). We also denote by $||\!\!-_{\mathcal{C}} \subseteq \mathcal{P}_f(\mathcal{A}) \times \mathcal{A}$ the set of all instances of these metarules. We are interested here (and later in section 10.1) in special subsets of $\mathcal{A}$ which express an ordinary weighted equation $(p, S, S')$ together with an admissible binary relation $\mathcal{R}$ of finite order (which is a *candidate* to be a $w$-$\bar{\psi}$-bisimulation w.r.t. $(S, S')$).

For every $p, n \in \mathbb{N}$, $\lambda \in \mathbb{N} - \{0\}$, $S, S' \in \mathsf{DRB}_{1,\lambda}\langle\langle\ V\ \rangle\rangle$, $\mathcal{R} \in \bar{\mathcal{B}}_n$, we use the notation

$$[p, S, S', \mathcal{R}] = \{(p + |u|, S \odot u, S' \odot u') \mid (u, u') \in \mathcal{R}\}. \tag{4.11}$$

One can check the following properties.

*Composition.* For every $p, n \in \mathbb{N}$, $\lambda \in \mathbb{N} - \{0\}$, $S, T \in \mathsf{DRB}_{1,\lambda}\langle\langle\ V\ \rangle\rangle$, $\mathcal{R}_1, \mathcal{R}_2 \in \bar{\mathcal{B}}_n$,

$$[p, S, S', \mathcal{R}_1] \cup [p, S', S'', \mathcal{R}_2] \ \overset{\langle * \rangle}{||\!\!-}_{\mathcal{C}} \ [p, S, S'', \mathcal{R}_1 \circ \mathcal{R}_2].$$

*Star.* For every $p, n \in \mathbb{N}$, $\lambda \in \mathbb{N} - \{0\}$, $S_1 \in \mathsf{DRB}_{1,1}\langle\langle\ V\ \rangle\rangle$, $S_1 \not\equiv \epsilon$, $(S_1, S) \in \mathsf{DRB}_{1,\lambda+1}\langle\langle\ V\ \rangle\rangle$, $T \in \mathsf{DRB}_{1,\lambda}\langle\langle\ V\ \rangle\rangle$, $\mathcal{R} \in \bar{\mathcal{B}}_n$,

$$[p, S_1 \cdot T + S, T, \mathcal{R}] \ \overset{\langle * \rangle}{|\!\!\vdash\!\!-}_{\mathcal{C}} [p, S_1^* \cdot S, T, \mathcal{R}^{\langle S_1, * \rangle}].$$

*Right-product.* For every $p, n \in \mathbb{N}$, $\delta, \lambda \in \mathbb{N} - \{0\}$, $S, S' \in \mathsf{DRB}_{1,\delta}\langle\langle\ V\ \rangle\rangle$, $T \in \mathsf{DRB}_{\delta,\lambda}\langle\langle\ V\ \rangle\rangle$, $\mathcal{R} \in \bar{\mathcal{B}}_n$,

$$[p, S, S', \mathcal{R}] \ \overset{\langle * \rangle}{|\!\!\vdash\!\!-}_{\mathcal{C}} [p, S \cdot T, S' \cdot T, \langle S | \mathcal{R} \rangle].$$

*Left-product.* For every $p, n \in \mathbb{N}$, $\delta, \lambda \in \mathbb{N} - \{0\}$, $S \in \mathsf{DRB}_{1,\delta}\langle\langle\ V\ \rangle\rangle$, $T, T' \in \mathsf{DRB}_{\delta,\lambda}\langle\langle\ V\ \rangle\rangle$, $\mathcal{R}_1, \ldots, \mathcal{R}_\delta \in \bar{\mathcal{B}}_n$,

$$\bigcup_{1 \le i \le \delta} [p, T_{i,*}, T'_{i,*}, \mathcal{R}_i] \ \overset{\langle * \rangle}{|\!\!\vdash\!\!-}_{\mathcal{C}} [p, S \cdot T, S \cdot T', \langle S, \mathcal{R} \rangle].$$

Given a subset $P \in \mathcal{P}_f(\mathcal{A})$, we define the *congruence closure* of $P$, denoted by $\mathrm{Cong}(P)$, to be the set

$$\mathrm{Cong}(P) = \{A \in \mathcal{A} \mid P \ \overset{\langle * \rangle}{|\!\!\vdash\!\!-}_{\mathcal{C}} \{A\}\}. \tag{4.12}$$

Also, for every integer $q \ge 0$ we define

$$\mathrm{Cong}_q(P) = \{A \in \mathcal{A} \mid P \ \overset{\langle q \rangle}{|\!\!\vdash\!\!-}_{\mathcal{C}} \{A\}\}. \tag{4.13}$$

## 5. Deterministic spaces.

### 5.1. Linear independence.

We adapt here the key idea of [23, 24] to bisimulation of vectors.

*Definitions.* Let $(W, \sim)$ be some structured alphabet. A vector $U = \sum_{i=1}^n \gamma_i \cdot U_i$ where $\vec{\gamma} \in \mathsf{DRB}_{1,n}\langle\langle\ W\ \rangle\rangle$, $U_i \in \mathsf{DRB}_{1,\lambda}\langle\langle\ W\ \rangle\rangle$ is called a *linear combination* of the $U_i$'s. We define as a *deterministic space* of rational vectors (d-space for short) any subset $\mathsf{V}$ of $\mathsf{DRB}_{1,\lambda}\langle\langle\ W\ \rangle\rangle$ which is closed under finite linear combinations. Given any set $\mathcal{G} = \{U_i \mid i \in I\} \subseteq \mathsf{DRB}_{1,\lambda}\langle\langle\ W\ \rangle\rangle$, one can check that the set $\mathsf{V}$ of all (finite) linear combinations of elements of $\mathcal{G}$ is a d-space (by Lemma 3.11) and that it is the smallest d-space containing $\mathcal{G}$. Therefore we call $\mathsf{V}$ the d-space *generated* by $\mathcal{G}$ and we call $\mathcal{G}$ a *generating set* of $\mathsf{V}$ (we note $\mathsf{V} = \mathsf{V}(\{U_i \mid i \in I\})$). (Similar definitions can be given for *families* of vectors.)

*Linear independence.* We now let $W = V$. Following an analogy with classical linear algebra, we now develop a notion corresponding to a kind of *linear independence* of the classes (mod $\sim$) of the given vectors. Let us extend the equivalence relation $\sim$ to d-spaces, as follows: if $\mathsf{V}_1, \mathsf{V}_2$ are d-spaces,

$$\mathsf{V}_1 \sim \mathsf{V}_2 \Leftrightarrow \forall i, j \in \{1, 2\}, \forall S \in \mathsf{V}_i, \exists S' \in \mathsf{V}_j, S \sim S'.$$

LEMMA 5.1. *Let $S_1, \ldots, S_j, \ldots, S_m \in \mathsf{DRB}_{1,\lambda}\langle\langle\ V\ \rangle\rangle$. The following are equivalent:*

(1) $\exists \vec{\alpha}, \vec{\beta} \in \mathsf{DRB}_{1,m}\langle\langle\ V\ \rangle\rangle$, $\vec{\alpha} \not\sim \vec{\beta}$, *such that*

$$\sum_{j=1}^m \alpha_j \cdot S_j \sim \sum_{j=1}^m \beta_j \cdot S_j.$$

(2) $\exists j_0 \in [1,m]$, $\exists \vec{\gamma} \in \mathsf{DRB}_{1,m}\langle\langle\ V\ \rangle\rangle$, $\vec{\gamma} \not\sim \epsilon^m_{j_0}$, *such that*

$$S_{j_0} \sim \sum_{j=1}^{m} \gamma_j \cdot S_j.$$

(3) $\exists j_0 \in [1,m]$, $\exists \vec{\gamma}' \in \mathsf{DRB}_{1,m}\langle\langle\ V\ \rangle\rangle$, $\gamma'_{j_0} \sim \emptyset$, *such that*

$$S_{j_0} \sim \sum_{j=1}^{m} \gamma'_j \cdot S_j.$$

(4) $\exists j_0 \in [1,m]$, *such that*

$$\mathsf{V}((S_j)_{1\le j\le m}) \sim \mathsf{V}((S_j)_{1\le j\le m, j\ne j_0}).$$

The equivalence between (1), (2), and (3) was first proved in [23, 24] in the case where the $S_j$'s are configurations $q_j\omega$, with the same $\omega$ and $\bar{\psi} = \mathrm{Id}_{X^*}$, and hence $\sim$ is just the language equivalence relation $\equiv$. This is the key idea around which we have developed the notion of d-spaces.

*Proof.* **(1) ⇒ (2):** Let us consider $\mathcal{R} \in \mathcal{B}_\infty(\vec{\alpha}\cdot S, \vec{\beta}\cdot S)$, $\nu = \mathrm{Div}(\vec{\alpha}, \vec{\beta})$, and

$$\begin{aligned}(u,v) = \min\{(w,w') \in \mathcal{R} \cap X^{\le\nu} \times X^{\le\nu} \mid \exists j \in [1,m],\\ (\vec{\alpha}\odot w = \epsilon^m_j) \Leftrightarrow (\vec{\beta}\odot w' \ne \epsilon^m_j)\}.\end{aligned} \tag{5.1}$$

Let us suppose, for example, that $\vec{\alpha}\odot u = \epsilon^m_{j_0}$ while $\vec{\beta}\odot v \ne \epsilon^m_{j_0}$ and let $\vec{\gamma} = \vec{\beta}\odot u$. As $(u,v) \in \mathcal{R} \in \mathcal{B}_\infty(\vec{\alpha}\cdot S, \vec{\beta}\cdot S)$,

$$(\vec{\alpha}\cdot S)\odot u \sim (\vec{\beta}\cdot S)\odot v. \tag{5.2}$$

Using Lemma 3.15 we obtain

$$(\vec{\alpha}\cdot S)\odot u = S_{j_0}. \tag{5.3}$$

Let us examine now the right-hand side of equality (5.2). Let $(u',v') \prec (u,v)$ with $|u'| = |v'|$. By condition (4) in Definition 3.21 $(u',v') \in \mathcal{R}$ (here is the main place where this condition (4) is used) and by minimality of $v$, $\vec{\beta}\odot v'$ is a unit iff $\vec{\alpha}\odot u'$ is a unit. But if $\vec{\alpha}\odot u'$ is a unit, then $\vec{\alpha}\odot u = \emptyset$, which is false. Hence $\vec{\beta}\odot v'$ is not a unit. Hence, $\forall v' \prec v$, $\vec{\beta}\odot v'$ is not a unit. By Lemma 3.15

$$(\vec{\beta}\cdot S)\odot v = (\vec{\beta}\odot v)\cdot S. \tag{5.4}$$

Let us plug equalities (5.3) and (5.4) into equivalence (5.2) and let us define $\vec{\gamma} = \vec{\beta}\odot v$. We obtain

$$S_{j_0} \sim \vec{\gamma}\cdot S, \qquad \vec{\gamma} \ne \epsilon^m_{j_0}.$$

**(2) ⇒ (3):**

$$S_{j_0} \sim \gamma_{j_0}\cdot S_{j_0} + \left(\sum_{j\ne j_0} \gamma_j \cdot S_j\right), \quad \gamma_{j_0} \ne \epsilon.$$

By Corollary 4.10, point (C1), we can deduce that

$$S_{j_0} \sim \sum_{j \neq j_0} \gamma_{j_0}^* \gamma_j \cdot S_j = \nabla_{j_0}^*(\gamma) \cdot S.$$

Taking $\gamma' = \nabla_{j_0}^*(\gamma)$ we obtain

$$S_{j_0} \sim \gamma' \cdot S, \text{ where } \gamma'_{j_0} = \emptyset.$$

**(3) ⇒ (4):** Let us denote by $\hat{S}$ the vector $(S_1, \ldots, S_{j_0-1}, \emptyset, S_{j_0+1} \ldots, S_m) \in \mathsf{DB}_{m,1}\langle\langle\, V\,\rangle\rangle$. If $T = \vec{\alpha} \cdot S$, then $T \sim (\vec{\alpha}\nabla_{j_0}\vec{\gamma'}) \cdot \hat{S}$.

**(4) ⇒ (1):** Let us suppose (4) is true for some integer $j_0$. The element $S_{j_0}$ is clearly equivalent (mod $\sim$) to two linear combinations of the $S_j$'s with nonequivalent vectors of coefficients (mod $\sim$). Hence (1) is true. □

**5.2. Triangulations.** Let $S_1, S_2, \ldots, S_d$ be a family of deterministic row-vectors over the structured alphabet $V$ (i.e., $S_i \in \mathsf{DRB}_{1,\lambda}\langle\langle\, V\,\rangle\rangle$, where $\lambda \in \mathbb{N} - \{0\}$). We recall that $V$ is the alphabet associated with some dpda $\mathcal{M}$ as defined in section 2.4.

Let us consider a sequence $\mathcal{S}$ of $n$ "weighted" linear equations:

$$(\mathcal{E}_i) : p_i, \sum_{j=1}^{d} \alpha_{i,j} S_j, \sum_{j=1}^{d} \beta_{i,j} S_j, \tag{5.5}$$

where $p_i \in \mathbb{N} - \{0\}$, and $A = (\alpha_{i,j})$, $B = (\beta_{i,j})$ are deterministic rational matrices of dimension $(n, d)$, with indices $m \leq i \leq m + n - 1$, $1 \leq j \leq d$.

For any weighted equation, $\mathcal{E} = (p, S, S')$, we recall that the "cost" of this equation is $H(\mathcal{E}) = p + 2 \cdot \mathrm{Div}(S, S')$.

Let us define an *oracle* on deterministic vectors as a mapping $\mathcal{O} : \bigcup_{\lambda \geq 1} \mathsf{DRB}_{1,\lambda}\langle\langle\, V\,\rangle\rangle \times \mathsf{DRB}_{1,\lambda}\langle\langle\, V\,\rangle\rangle \to \mathcal{P}(X^* \times X^*)$ such that

$$\forall (S, S') \in \mathsf{DRB}_{1,\lambda}\langle\langle\, V\,\rangle\rangle \times \mathsf{DRB}_{1,\lambda}\langle\langle\, V\,\rangle\rangle, \quad S \sim S' \Rightarrow \mathcal{O}(S, S') \in \mathcal{B}_\infty(S, S').$$

In other words, an oracle is a *choice* of $w$-$\bar{\psi}$-bisimulation for every pair of equivalent vectors (modulo $\sim$). Let us denote by $\Omega$ the set of all oracles. Let us fix an oracle $\mathcal{O}$ throughout this section.

We associate to every system (5.5) another equation, $\mathrm{INV}^{(\mathcal{O})}(\mathcal{S})$, which "translates the equations of $\mathcal{S}$ into equations over the coefficients $(\alpha_{i,j}, \beta_{i,j})$ only."[2] The general idea of the construction of $\mathrm{INV}^{(\mathcal{O})}$ consists in iterating the transformation used in the proof of (1) ⇒ (2) ⇒ (3) in Lemma 5.1, i.e., the classical idea of *triangulating* a system of linear equations. Of course we must deal with the weights and relate the construction with the deduction system $\mathcal{B}_0$.

We assume here that

$$\forall j \in [1, d], \quad S_j \neq \emptyset^\lambda. \tag{5.6}$$

Let us define $\mathrm{INV}^{(\mathcal{O})}(\mathcal{S}), \mathrm{W}^{(\mathcal{O})}(\mathcal{S}) \in \mathbb{N} \cup \{\perp\}$, $\mathrm{D}^{(\mathcal{O})}(\mathcal{S}) \in \mathbb{N}$ by induction on $n$. $\mathrm{W}^{(\mathcal{O})}(\mathcal{S})$ is the *weight* of $\mathcal{S}$. $\mathrm{D}^{(\mathcal{O})}(\mathcal{S})$ is the *weak codimension* of $\mathcal{S}$.

[2]The function INV defined in [34] was an "elaborated version" of the *inverse* systems defined in [23, 24] in the case of a single equation. We consider here a *relativization* of this notion to some oracle $\mathcal{O}$.

*Case* 1. $\alpha_{m,*} \sim \beta_{m,*}$.

$$\mathrm{INV}^{(\mathcal{O})}(\mathcal{S}) = (\mathrm{W}^{(\mathcal{O})}(\mathcal{S}), \alpha_{m,*}, \beta_{m,*}), \quad \mathrm{W}^{(\mathcal{O})}(\mathcal{S}) = p_m - 1, \quad \mathrm{D}^{(\mathcal{O})}(\mathcal{S}) = 0.$$

*Case* 2. $\alpha_{m,*} \not\sim \beta_{m,*}$, $n \geq 2$, $p_{m+1} - p_m \geq 2 \cdot \mathrm{Div}(\alpha_{m,*}, \beta_{m,*}) + 1$. Let us consider $\mathcal{R} = \mathcal{O}(\sum_{j=1}^{d} \alpha_{m,j} S_j, \sum_{j=1}^{d} \beta_{m,j} S_j)$, $\nu = \mathrm{Div}(\alpha_{m,*}, \beta_{m,*})$, and

$$(u, u') = \min\{(v, v') \in \mathcal{R} \cap X^{\leq \nu} \times X^{\leq \nu} \mid \exists j \in [1, d], \\ (\alpha_{m,*} \odot v = \epsilon_j^{\lambda}) \Leftrightarrow (\beta_{m,*} \odot v' \neq \epsilon_j^{\lambda})\}. \tag{5.7}$$

Let us consider the integer $j_0 \in [1, d]$ such that $(\alpha_{m,*} \odot u = \epsilon_{j_0}^{\lambda}) \Leftrightarrow (\beta_{m,*} \odot u' \neq \epsilon_{j_0}^{\lambda})$.

*Subcase* 1. $\alpha_{m,j_0} \odot u = \varepsilon$, $\beta_{m,j_0} \odot u' \neq \varepsilon$. Let us consider the equation

$$(\mathcal{E}'_m) : p_m + 2 \cdot |u|,\ S_{j_0},\ \sum_{\substack{j=1 \\ j \neq j_0}}^{d} (\beta_{m,j_0} \odot u')^* (\beta_{m,j} \odot u') S_j$$

and define a new system of weighted equations $\mathcal{S}' = (\mathcal{E}'_i)_{m+1 \leq i \leq m+n-1}$ by

$$(\mathcal{E}'_i) : p_i,\ \sum_{j \neq j_0} [(\alpha_{i,j} + \alpha_{i,j_0} (\beta_{m,j_0} \odot u')^* (\beta_{m,j} \odot u')] S_j, \\ \sum_{j \neq j_0} [(\beta_{i,j} + \beta_{i,j_0} (\beta_{m,j_0} \odot u')^* (\beta_{m,j} \odot u')] S_j, \tag{5.8}$$

where the above equation is seen as an equation between two linear combinations of the $S_i$'s, $1 \leq i \leq d$, where the $j_0$th coefficient is $\emptyset$ on both sides. We then define

$$\mathrm{INV}^{(\mathcal{O})}(\mathcal{S}) = \mathrm{INV}^{(\mathcal{O})}(\mathcal{S}'), \quad \mathrm{W}^{(\mathcal{O})}(\mathcal{S}) = \mathrm{W}^{(\mathcal{O})}(\mathcal{S}'), \quad \mathrm{D}^{(\mathcal{O})}(\mathcal{S}) = \mathrm{D}^{(\mathcal{O})}(\mathcal{S}') + 1.$$

*Subcase* 2. $\alpha_{m,j_0} \odot u \neq \varepsilon$, $\beta_{m,j_0} \odot u' = \varepsilon$ (analogous to Subcase 1).

*Case* 3. $\alpha_{m,*} \not\sim \beta_{m,*}$, $n = 1$. We then define

$$\mathrm{INV}^{(\mathcal{O})}(\mathcal{S}) = \perp, \quad \mathrm{W}^{(\mathcal{O})}(\mathcal{S}) = \perp, \quad \mathrm{D}^{(\mathcal{O})}(\mathcal{S}) = 0,$$

where $\perp$ is a special symbol which can be understood as meaning "undefined."

*Case* 4. $\alpha_{m,*} \not\sim \beta_{m,*}$, $n \geq 2$, $p_{m+1} - p_m \leq 2 \cdot \mathrm{Div}(\alpha_{m,*}, \beta_{m,*})$. We then define

$$\mathrm{INV}^{(\mathcal{O})}(\mathcal{S}) = \perp, \quad \mathrm{W}^{(\mathcal{O})}(\mathcal{S}) = \perp, \quad \mathrm{D}^{(\mathcal{O})}(\mathcal{S}) = 0.$$

LEMMA 5.2. *Let $\mathcal{S}$ be a system of weighted linear equations with deterministic rational coefficients. If $\mathrm{INV}^{(\mathcal{O})}(\mathcal{S}) \neq \perp$, then $\mathrm{INV}^{(\mathcal{O})}(\mathcal{S})$ is a weighted linear equation with deterministic rational coefficients.*

*Proof.* The proof follows from Lemmas 3.18 and 3.19 and the formula defining $\mathcal{S}'$ from $\mathcal{S}$. □

From now on, and up to the end of this section, we simply write "linear equation" to mean "weighted linear equations with deterministic rational coefficients."

LEMMA 5.3. *Let $\mathcal{S}$ be a system of linear equations. If $\mathrm{INV}^{(\mathcal{O})}(\mathcal{S}) \neq \perp$, then*

(1) $\{\mathrm{INV}^{(\mathcal{O})}(\mathcal{S})\} \cup \{\mathcal{E}_i \mid m \leq i \leq m + \mathrm{D}^{(\mathcal{O})}(\mathcal{S}) - 1\} \mid\!\!\!-\!\!- \mathcal{E}_{m+\mathrm{D}^{(\mathcal{O})}(\mathcal{S})}$;

(2) $\min\{H(\mathcal{E}_i) \mid m \leq i \leq m + \mathrm{D}^{(\mathcal{O})}(\mathcal{S})\} = \infty \Longrightarrow H(\mathrm{INV}^{(\mathcal{O})}(\mathcal{S})) = \infty$.

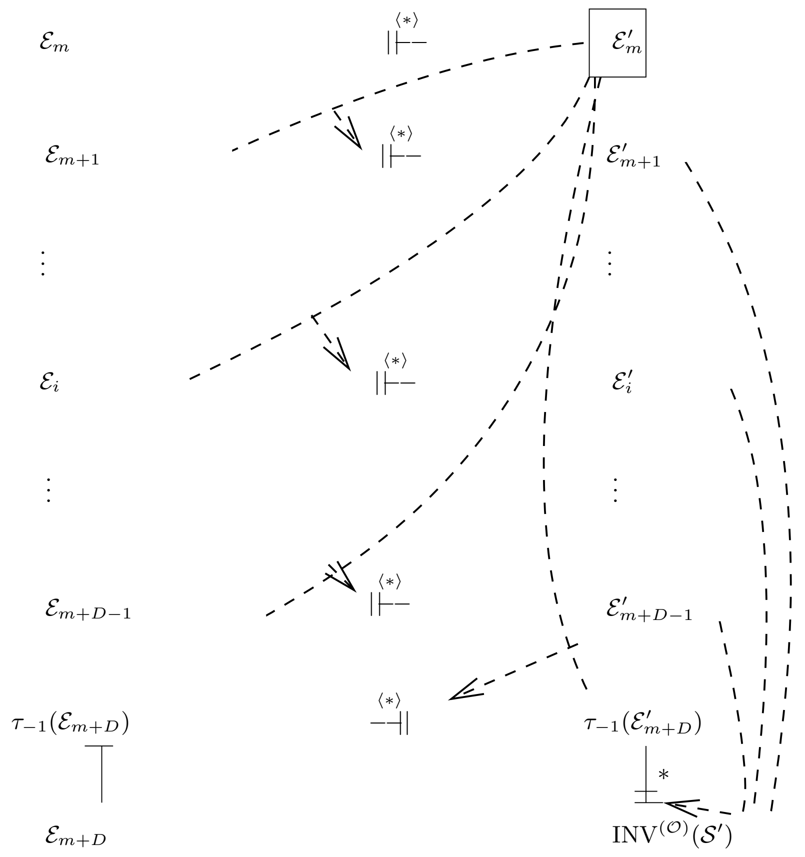


FIG. 2. *Proof of Lemma* 5.2.

*Proof.* See in Figure 2 the "graph of the deductions" we use for proving point (1). Let us prove by induction on $\mathrm{D}^{(\mathcal{O})}(\mathcal{S})$ the following strengthened version of point (1):

$$\{\mathrm{INV}^{(\mathcal{O})}(\mathcal{S})\} \cup \{\mathcal{E}_i \mid m \leq i \leq m + \mathrm{D}^{(\mathcal{O})}(\mathcal{S}) - 1\} \overset{\langle * \rangle}{||\!\!-\!\!-} \tau_{-1}(\mathcal{E}_{m+\mathrm{D}^{(\mathcal{O})}(\mathcal{S})}), \tag{5.9}$$

where, for every integer $k \in \mathbb{Z}$, $\tau_k : \{(p, S, S') \in \mathcal{A} \mid p \geq -k\} \to \mathcal{A}$ is the *translation* map on the weights $\tau_k(p, S, S') = (p + k, S, S')$.

**If $\mathbf{D}^{(\mathcal{O})}(\mathcal{S}) = 0$:** As $\mathrm{INV}^{(\mathcal{O})}(\mathcal{S}) \neq \perp$, $\mathcal{S}$ must fulfill the hypothesis of Case 1.

$$\mathcal{E}_m = \left(p_m, \sum_{j=1}^{d} \alpha_{m,j} S_j, \sum_{j=1}^{d} \beta_{m,j} S_j\right) = \mathcal{E}_{m+\mathrm{D}^{(\mathcal{O})}(\mathcal{S})},$$

$$\mathrm{INV}^{(\mathcal{O})}(\mathcal{S}) = (p_m - 1, \alpha_{m,*}, \beta_{m,*}).$$

Using rule (R7) we obtain

$$\mathrm{INV}^{(\mathcal{O})}(\mathcal{S}) \overset{\langle * \rangle}{||\!\!-\!\!-} \left(p_m - 1, \sum_{j=1}^{d} \alpha_{m,j} S_j, \sum_{j=1}^{d} \beta_{m,j} S_j\right) = \tau_{-1}(\mathcal{E}_m).$$

**If $\mathrm{D}^{(\mathcal{O})}(\mathcal{S}) = n+1$, $n \geq 0$:** $\mathcal{S}$ must fulfill Case 2.

• Suppose Case 2, Subcase 1 occurs.

As the relation $\mathcal{R}$ used in the construction of $\mathcal{E}'_m$ from $\mathcal{E}_m$ is a $w$-$\bar{\psi}$-bisimulation w.r.t. the pair of sides of equation $\mathcal{E}_m$, using (R5) and then (R6) (this is possible because $\beta_{m,j_0} \odot u' \neq \epsilon$), we obtain a deduction:

$$\mathcal{E}_m \overset{\langle 2\cdot|u|+1\rangle}{||\!\!-\!\!-} \mathcal{E}'_m. \tag{5.10}$$

Using (R2), (R8) we get that, for every $i \in [m+1, m+\mathrm{D}^{(\mathcal{O})}(\mathcal{S})]$,

$$\{\mathcal{E}_i, \mathcal{E}'_m\} \overset{\langle * \rangle}{||\!\!-\!\!-}$$

$$\left(\max\{p_i, p_m + 2\,|u|\}, \sum_{j\neq j_0}(\alpha_{i,j} + \alpha_{i,j_0}(\beta_{m,j}\odot u'))S_j, \sum_{j\neq j_0}(\beta_{i,j} + \beta_{i,j_0}(\beta_{m,j}\odot u'))S_j\right)$$

but the hypothesis of Case 2 implies that $\max\{p_{m+1}, p_m + 2 \mid u \mid\} = p_{m+1}$, and the fact that $\mathrm{INV}^{(\mathcal{O})}(\mathcal{S}')$ is defined implies that $\forall i \in [m+1, m+\mathrm{D}^{(\mathcal{O})}(\mathcal{S})]$, $p_i \geq p_{m+1}$; hence, $\max\{p_i, p_m + 2 \mid u \mid\} = p_i$ and the right-hand side of the above deduction is exactly $\mathcal{E}'_i$. Therefore,

$$\forall i \in [m+1, m+\mathrm{D}^{(\mathcal{O})}(\mathcal{S})], \quad \{\mathcal{E}_i, \mathcal{E}'_m\} \overset{\langle * \rangle}{||\!\!-\!\!-} \mathcal{E}'_i. \tag{5.11}$$

Using deductions (5.10) and (5.11), we obtain that

$$\{\mathcal{E}_i \mid m \leq i \leq m+\mathrm{D}^{(\mathcal{O})}(\mathcal{S})-1\} \overset{\langle * \rangle}{||\!\!-\!\!-} \{\mathcal{E}'_i \mid m \leq i \leq m+\mathrm{D}^{(\mathcal{O})}(\mathcal{S})-1\}. \tag{5.12}$$

By the induction hypothesis,

$$\mathrm{INV}^{(\mathcal{O})}(\mathcal{S}') \cup \{\mathcal{E}'_i \mid m+1 \leq i \leq m+1+\mathrm{D}^{(\mathcal{O})}(\mathcal{S}')-1\} \overset{\langle * \rangle}{||\!\!-\!\!-} \tau_{-1}(\mathcal{E}'_{m+1+\mathrm{D}^{(\mathcal{O})}(\mathcal{S}')}),$$

which is equivalent to

$$\mathrm{INV}^{(\mathcal{O})}(\mathcal{S}) \cup \{\mathcal{E}'_i \mid m+1 \leq i \leq m+\mathrm{D}^{(\mathcal{O})}(\mathcal{S})-1\} \overset{\langle * \rangle}{||\!\!-\!\!-} \tau_{-1}(\mathcal{E}'_{m+\mathrm{D}^{(\mathcal{O})}(\mathcal{S})}). \tag{5.13}$$

As $p_m + 2\cdot|u| \leq p_{m+1} - 1 \leq p_{m+\mathrm{D}^{(\mathcal{O})}(\mathcal{S})} - 1$, we have also the following inverse deduction (which is similar to deduction (5.11)):

$$\{\mathcal{E}'_m, \tau_{-1}(\mathcal{E}'_{m+\mathrm{D}^{(\mathcal{O})}(\mathcal{S})})\} \overset{\langle * \rangle}{||\!\!-\!\!-} \tau_{-1}(\mathcal{E}_{m+\mathrm{D}^{(\mathcal{O})}(\mathcal{S})}). \tag{5.14}$$

Combining deductions (5.12), (5.13), and (5.14), we have proved (5.9). Using rule (R0), this last deduction leads to point (1) of the lemma.

• Suppose now that Case 2, Subcase 2 occurs.

This case can be treated in the same way as Subcase 1, by simply exchanging the roles of $\alpha, \beta$.

Let us prove statement (2) of the lemma.

We prove by induction on $\mathrm{D}^{(\mathcal{O})}(\mathcal{S})$ the statement

$$\min\{H(\mathcal{E}_i) \mid m \leq i \leq m + \mathrm{D}^{(\mathcal{O})}(\mathcal{S})\} = \infty \Longrightarrow H(\mathrm{INV}^{(\mathcal{O})}(\mathcal{S})) = \infty. \tag{5.15}$$

**If $\mathbf{D}^{(\mathcal{O})}(\mathcal{S}) = 0$:** As $\mathrm{INV}^{(\mathcal{O})}(\mathcal{S}) \neq \perp$, Case 1 must occur. $\alpha_{m,*} \sim \beta_{m,*}$ implies that $H(\mathrm{INV}^{(\mathcal{O})}(\mathcal{S})) = \infty$, and hence the statement is true.

**If $\mathbf{D}^{(\mathcal{O})}(\mathcal{S}) = p + 1$, $p \geq 0$:** As $\mathrm{D}^{(\mathcal{O})}(\mathcal{S}) \geq 1$ and $\mathrm{INV}^{(\mathcal{O})}(\mathcal{S}) \neq \perp$, Case 2 must occur.

Using deductions (5.10) and (5.11) established above we obtain that

$$\{\mathcal{E}_i \mid m \leq i \leq m + \mathrm{D}^{(\mathcal{O})}(\mathcal{S})\} \overset{\langle * \rangle}{||\!\!-\!\!-} \{\mathcal{E}'_i \mid m + 1 \leq i \leq m + 1 + \mathrm{D}^{(\mathcal{O})}(\mathcal{S}')\},$$

which proves that

$$\min\{H(\mathcal{E}_i) \mid m \leq i \leq m + \mathrm{D}^{(\mathcal{O})}(\mathcal{S})\} \leq \min\{H(\mathcal{E}'_i) \mid m + 1 \leq i \leq m + 1 + \mathrm{D}^{(\mathcal{O})}(\mathcal{S}')\}. \tag{5.16}$$

Since $\mathrm{D}^{(\mathcal{O})}(\mathcal{S}') = \mathrm{D}^{(\mathcal{O})}(\mathcal{S}) - 1$, we can use the induction hypothesis

$$\min\{H(\mathcal{E}'_i) \mid m + 1 \leq i \leq m + 1 + D(\mathcal{S}')\} = \infty \Longrightarrow H(\mathrm{INV}^{(\mathcal{O})}(\mathcal{S}')) = \infty. \tag{5.17}$$

Since $\mathrm{INV}^{(\mathcal{O})}(\mathcal{S}) = \mathrm{INV}^{(\mathcal{O})}(\mathcal{S}')$, (5.16), (5.17) imply statement (5.15). $\square$

LEMMA 5.4. *Let $\mathcal{S}$ be a system of linear equations satisfying the hypothesis of Case 2. Then, $\forall i \in [m+1, m+n-1]$,*

$$\| \alpha'_{i,*} \| \leq \| \alpha_{i,*} \| + \| \beta_{m,*} \| + K_0 \mid u \mid,$$
$$\| \beta'_{i,*} \| \leq \| \beta_{i,*} \| + \| \beta_{m,*} \| + K_0 \mid u \mid .$$

*Proof.* The formula defining $\mathcal{S}'$ from $\mathcal{S}$ shows that

$$\alpha'_{i,*} = \alpha_{i,*} \nabla_{j_0}(\nabla^*_{j_0}(\beta_{m,*} \odot u')),$$
$$\beta'_{i,*} = \beta_{i,*} \nabla_{j_0}(\nabla^*_{j_0}(\beta_{m,*} \odot u')).$$

From these equalities and Lemmas 3.18, 3.19, and 3.13, the inequalities on the norm follow. $\square$

Let us consider the function $F$ defined by

$$F(d, n) = \max\{\mathrm{Div}(A, B) \mid A, B \in \mathsf{DRB}_{1,d}\langle\langle V \rangle\rangle, \| A \| \leq n, \| B \| \leq n, A \not\sim B\}.$$

For every integer parameter $K_0, K_1, K_2, K_3, K_4 \in \mathbb{N} - \{0\}$, we define integer sequences $(\delta_i, \ell_i, L_i, s_i, S_i, \Sigma_i)_{m \leq i \leq m+n-1}$ by

(5.18) $$\delta_m = 0,\ \ell_m = 0,\ L_m = K_2,\ s_m = K_3 \cdot K_2 + K_4,\ S_m = 0,\ \Sigma_m = 0,$$

(5.19) $$\begin{cases} \delta_{i+1} = 2 \cdot F(d, s_i + \Sigma_i) + 1, \\ \ell_{i+1} = 2 \cdot \delta_{i+1} + 3, \\ L_{i+1} = K_1 \cdot (L_i + \ell_{i+1}) + K_2, \\ s_{i+1} = K_3 \cdot L_{i+1} + K_4, \\ S_{i+1} = s_i + \Sigma_i + K_0 F(d, s_i + \Sigma_i), \\ \Sigma_{i+1} = \Sigma_i + S_{i+1} \end{cases}$$

for $m \leq i \leq m + n - 2$.

These sequences are intended to have the following meanings when $K_0, K_1, K_2, K_3, K_4$ are chosen to be the constants defined in section 6 and the equations $(\mathcal{E}_i)$ are labeling nodes of a B-stacking sequence (see section 8.2):

- $\delta_{i+1} \leq$ increase of weight between $\mathcal{E}_i, \mathcal{E}_{i+1}$.
- $\ell_{i+1} \geq$ increase of depth between $\mathcal{E}_i, \mathcal{E}_{i+1}$.
- $L_{i+1} \geq$ increase of depth between $\mathcal{E}_m, \mathcal{E}_{i+1}$.
- $s_{i+1} \geq$ size of the coefficients of $\mathcal{E}_{i+1}$.
- $S_{i+1} \geq$ size of the coefficients of $\mathcal{E}_{i+1}^{(i+1-m)}$ (these systems are introduced below in the proof of Lemma 5.5).
- $\Sigma_{i+1} \geq$ increase of the coefficients between $\mathcal{E}_k^{(i-m)}$, $\mathcal{E}_k^{(i+1-m)}$ (for $k \geq i + 1$).

For every linear equation $\mathcal{E} = (p, \sum_{j=1}^{d} \alpha_j S_j, \sum_{j=1}^{d} \beta_j S_j)$, we define

$$||| \mathcal{E} ||| = \max\{\| (\alpha_1, \ldots, \alpha_d) \|, \| (\beta_1, \ldots, \beta_d) \|\}.$$

LEMMA 5.5. *Let $\mathcal{S} = (\mathcal{E}_i)_{m \leq i \leq m+d-1}$ be a system of $d$ linear equations such that $H(\mathcal{E}_i) = \infty$ (for every $i$) and*

(1) *$\forall i \in [m, m + d - 1]$, $||| \mathcal{E}_i ||| \leq s_i$,*
(2) *$\forall i \in [m, m + d - 2]$, $\mathrm{W}(\mathcal{E}_{i+1}) - \mathrm{W}(\mathcal{E}_i) \geq \delta_{i+1}$.*

*Then*

(3) *$\mathrm{INV}^{(\mathcal{O})}(\mathcal{S}) \neq \perp$,*
(4) *$\mathrm{D}^{(\mathcal{O})}(\mathcal{S}) \leq d - 1$,*
(5) *$||| \mathrm{INV}^{(\mathcal{O})}(\mathcal{S}) ||| \leq \Sigma_{m+\mathrm{D}^{(\mathcal{O})}(\mathcal{S})} + s_{m+\mathrm{D}^{(\mathcal{O})}(\mathcal{S})}$.*

*Proof.* (Figure 3 might help the reader to follow the definitions below.) Let us define a sequence of systems $\mathcal{S}^{(i-m)} = (\mathcal{E}_k^{(i-m)})_{m \leq i \leq k \leq m+d-1}$, where $i \in [m, m + \mathrm{D}^{(\mathcal{O})}(\mathcal{S})]$; by induction

- $\mathcal{E}_k^{(0)} = \mathcal{E}_k$ for $m \leq k \leq m + d - 1$;
- if Case 1 or Case 3 or Case 4 is realized, $\mathrm{D}^{(\mathcal{O})}(\mathcal{S}) = 0$; hence $\mathcal{S}^{(i-m)}$ is well-defined for $m \leq i \leq m + \mathrm{D}^{(\mathcal{O})}(\mathcal{S})$;
- if Case 2 is realized, then we set, $\forall i \geq m + 1$, $\mathcal{E}_k^{(i-m)} = (\mathcal{E}'_k)^{(i-m-1)}$ for $m + 1 \leq k \leq m + d - 1$.

Let us prove by induction on $i \in [m, m + \mathrm{D}^{(\mathcal{O})}(\mathcal{S})]$ that, $\forall k \in [i, m + d - 1]$,

(5.20) $$||| \mathcal{E}_k^{(i-m)} ||| \leq s_k + \Sigma_i.$$

***i* = *m*:** In this case

$$||| \mathcal{E}_k^{(i-m)} ||| = ||| \mathcal{E}_k ||| \leq s_k = s_k + \Sigma_m.$$

**$i + 1 \leq m + \mathrm{D}^{(\mathcal{O})}(\mathcal{S})$:** In this case, by Lemma 5.4,

$$||| \mathcal{E}_k^{(i+1-m)} ||| \leq ||| \mathcal{E}_k^{(i-m)} ||| + ||| \mathcal{E}_i^{(i-m)} ||| + K_0 \mid u \mid,$$

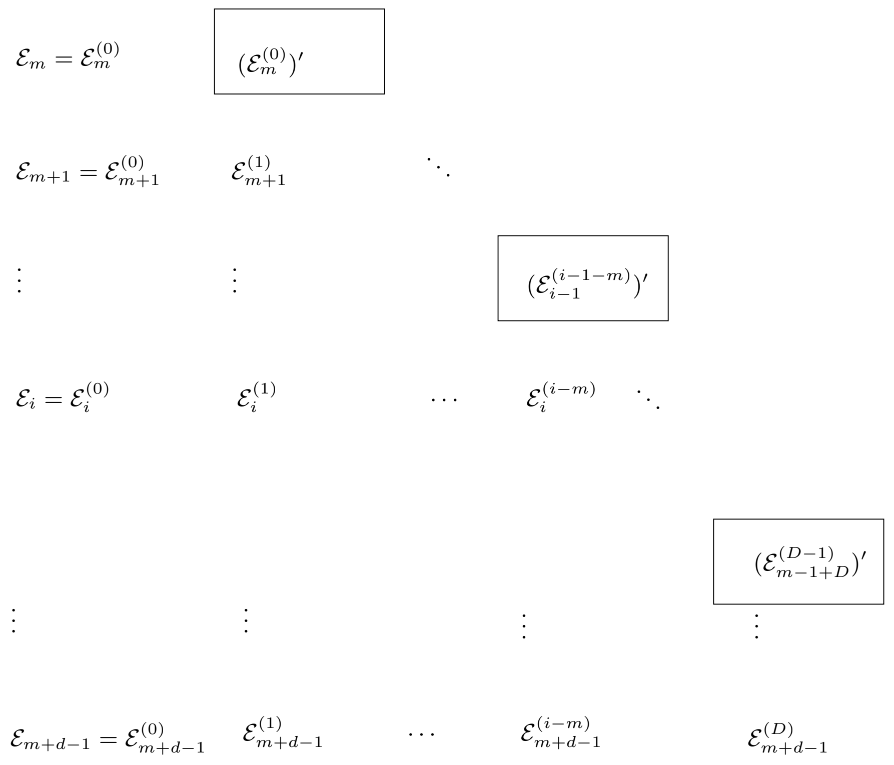

$\mathcal{E}_m = \mathcal{E}_m^{(0)}$
$(\mathcal{E}_m^{(0)})'$
$\mathcal{E}_{m+1} = \mathcal{E}_{m+1}^{(0)}$
$\mathcal{E}_{m+1}^{(1)}$
$(\mathcal{E}_{i-1}^{(i-1-m)})'$
$\mathcal{E}_i = \mathcal{E}_i^{(0)}$
$\mathcal{E}_i^{(1)}$
$\mathcal{E}_i^{(i-m)}$
$(\mathcal{E}_{m-1+D}^{(D-1)})'$
$\mathcal{E}_{m+d-1} = \mathcal{E}_{m+d-1}^{(0)}$
$\mathcal{E}_{m+d-1}^{(1)}$
$\mathcal{E}_{m+d-1}^{(i-m)}$
$\mathcal{E}_{m+d-1}^{(D)}$

FIG. 3. *Proof of Lemma* 5.4.

where $\mathcal{R} = \mathcal{O}(\sum_{j=1}^{d} \alpha_{i,j}^{(i-m)} S_j, \sum_{j=1}^{d} \beta_{i,j}^{(i-m)} S_j)$, $\nu = \mathrm{Div}(\alpha_{i,*}^{(i-m)}, \beta_{i,*}^{(i-m)})$, and

$$(u, u') = \min\{(v, v') \in \mathcal{R} \cap X^{\leq \nu} \times X^{\leq \nu} \mid \exists j \in [1, d],$$
$$(\alpha_{i,*}^{(i-m)} \odot v = \epsilon_j^{\lambda}) \Leftrightarrow (\beta_{i,*}^{(i-m)} \odot v' \neq \epsilon_j^{\lambda})\}.$$

By definition of $F$ and the induction hypothesis,

$$\mid u \mid \leq F(d, \||\, \mathcal{E}_i^{(i-m)} \,\||) \leq F(d, s_i + \Sigma_i).$$

Hence

$$\begin{aligned} \||\, \mathcal{E}_k^{(i+1-m)} \,\|| \leq (s_k + \Sigma_i) + (s_i + \Sigma_i) + K_0 F(d, s_i + \Sigma_i) &= (s_k + \Sigma_i) + S_{i+1} \\ &= s_k + \Sigma_{i+1}. \end{aligned}$$

Let us note that $\mathrm{D}^{(\mathcal{O})}(\mathcal{S})$ is always an integer and that this proof is valid for $m \leq i \leq m + \mathrm{D}^{(\mathcal{O})}(\mathcal{S})$, $i \leq k \leq m + d - 1$.

We now prove that $\mathrm{INV}^{(\mathcal{O})}(\mathcal{S}) \neq \perp$. Let us consider the system

$$(\mathcal{E}_k^{(\mathrm{D}^{(\mathcal{O})}(\mathcal{S}))})_{m+\mathrm{D}^{(\mathcal{O})}(\mathcal{S}) \leq k \leq m+d-1}.$$

**If $\mathbf{D^{(\mathcal{O})}(\mathcal{S}) = d-1}$:** $(\mathcal{E}^{(\mathrm{D}^{(\mathcal{O})}(\mathcal{S}))})$ fulfills either Case 1 or Case 3 of the definition of $\mathrm{INV}^{(\mathcal{O})}$ (just because this system consists of a single equation).

Using the successive deductions (5.10), (5.11) established in the proof of Lemma 5.3 we get that

$$\{\mathcal{E}_i \mid m \leq i \leq m+d-1\} \overset{\langle * \rangle}{||\!\!-\!\!-} \{\mathcal{E}^{(d-1)}_{m+d-1}\}.$$

Using now the hypothesis that $H(\mathcal{E}_i) = \infty$ (for $m \leq i \leq m+d-1$), we obtain

$$H(\mathcal{E}^{(d-1)}_{m+d-1}) = \infty. \tag{5.21}$$

For any system of equations $\mathcal{S}$, let us define the *support* of the system as

$$\operatorname{supp}(\mathcal{S}) = \left\{ j \in [1,d] \;\middle|\; \sum_{i=m}^{m+n-1} \alpha_{i,j} + \beta_{i,j} \neq \emptyset \right\}.$$

Let us consider $\delta = \operatorname{Card}(\operatorname{supp}(\mathcal{S}^{(d-1)})$. One can prove by induction on $i$ that

$$\operatorname{Card}(\operatorname{supp}(\mathcal{S}^{(i-m)}) \leq d - i + m,$$

and hence

$$\delta = \operatorname{Card}(\operatorname{supp}(\mathcal{S}^{(d-1)}) \leq d - (d-1) = 1.$$

- If $\delta = 1$, $\operatorname{supp}(\mathcal{S}^{(d-1)}) = \{j_0\}$ for some $j_0 \in [1,d]$.
  By Corollary 4.10, point (C3), and by hypothesis (5.6), the implication

  $$[(\alpha^{(d-1)}_{m+d-1,j_0} S_{j_0} \sim \beta^{(d-1)}_{m+d-1,j_0} S_{j_0}) \Longrightarrow \alpha^{(d-1)}_{m+d-1,j_0} \sim \beta^{(d-1)}_{m+d-1,j_0}]$$

  holds. Hence, by (5.21), $\alpha^{(d-1)}_{m+d-1,j_0} \sim \beta^{(d-1)}_{m+d-1,j_0}$; i.e., $\mathcal{S}^{(d-1)}$ fulfills Case 1, so that

  $$\mathrm{INV}^{(\mathcal{O})}(\mathcal{S}) = \mathrm{INV}^{(\mathcal{O})}(\mathcal{S}^{(d-1)}) \neq \perp.$$

- If $\delta = 0$, $\operatorname{supp}(\mathcal{S}) = \emptyset$.
  Then $\alpha^{(d-1)}_{m+d-1,*} = \beta^{(d-1)}_{m+d-1,*} = \emptyset^d$. Here also $\mathcal{S}^{(d-1)}$ fulfills Case 1.

**If $\mathbf{D}^{(\mathcal{O})}(\mathcal{S}) < d-1$:** By the hypothesis,

$$\begin{aligned}\mathrm{W}(\mathcal{E}_{m+\mathrm{D}^{(\mathcal{O})}(\mathcal{S})+1}) - \mathrm{W}(\mathcal{E}_{m+\mathrm{D}^{(\mathcal{O})}(\mathcal{S})}) &\geq \delta_{m+\mathrm{D}^{(\mathcal{O})}(\mathcal{S})+1} \\ &= 2F(d, s_{m+\mathrm{D}^{(\mathcal{O})}(\mathcal{S})} + \Sigma_{m+\mathrm{D}^{(\mathcal{O})}(\mathcal{S})}) + 1.\end{aligned}$$

If $\alpha^{\mathrm{D}^{(\mathcal{O})}(\mathcal{S})}_{m+\mathrm{D}^{(\mathcal{O})}(\mathcal{S}),*} \sim \beta^{\mathrm{D}^{(\mathcal{O})}(\mathcal{S})}_{m+\mathrm{D}^{(\mathcal{O})}(\mathcal{S}),*}$, then $\mathcal{E}^{(\mathrm{D}^{(\mathcal{O})}(\mathcal{S}))}_{m+\mathrm{D}^{(\mathcal{O})}(\mathcal{S})}$ fulfills Case 1 of the definition of $\mathrm{INV}^{(\mathcal{O})}$; hence $\mathrm{INV}^{(\mathcal{O})}(\mathcal{S}) \neq \perp$.

Otherwise, let us consider

$$\mathcal{R} = \mathcal{O}\left( \sum_{j=1}^{d} \alpha^{(\mathrm{D}^{(\mathcal{O})}(\mathcal{S}))}_{m+\mathrm{D}^{(\mathcal{O})}(\mathcal{S}),j} S_j, \sum_{j=1}^{d} \beta^{(\mathrm{D}^{(\mathcal{O})}(\mathcal{S}))}_{m+\mathrm{D}^{(\mathcal{O})}(\mathcal{S}),j} S_j \right),$$

$$\nu = \operatorname{Div}\left( \alpha^{(\mathrm{D}^{(\mathcal{O})}(\mathcal{S}))}_{m+\mathrm{D}^{(\mathcal{O})}(\mathcal{S}),*}, \beta^{(\mathrm{D}^{(\mathcal{O})}(\mathcal{S}))}_{m+\mathrm{D}^{(\mathcal{O})}(\mathcal{S}),*} \right), \text{ and}$$

$$(u,u') = \min\left\{ (v,v') \in \mathcal{R} \cap X^{\leq \nu} \times X^{\leq \nu} \;\middle|\; \exists j \in [1,d], \left( \alpha^{(\mathrm{D}^{(\mathcal{O})}(\mathcal{S}))}_{m+\mathrm{D}^{(\mathcal{O})}(\mathcal{S}),*} \odot v = \epsilon^{\lambda}_{j} \right) \Leftrightarrow \left( \beta^{(\mathrm{D}^{(\mathcal{O})}(\mathcal{S}))}_{m+\mathrm{D}^{(\mathcal{O})}(\mathcal{S}),*} \odot v' \neq \epsilon^{\lambda}_{j} \right) \right\}.$$

By the definition of $F$ and inequality (5.20),

$$| u | \leq F\left(d, \left\| \left\| \mathcal{E}^{(\mathrm{D}^{(\mathcal{O})}(\mathcal{S}))}_{m+\mathrm{D}^{(\mathcal{O})}(\mathcal{S})} \right\| \right\|\right) \leq F(d, s_{m+\mathrm{D}^{(\mathcal{O})}(\mathcal{S})} + \Sigma_{m+\mathrm{D}^{(\mathcal{O})}(\mathcal{S})}).$$

Hence $p_{m+\mathrm{D}^{(\mathcal{O})}(\mathcal{S})+1} - p_{m+\mathrm{D}^{(\mathcal{O})}(\mathcal{S})} \geq 2|u| + 1$; i.e., the hypothesis of Case 2 is realized. This proves that $\mathrm{D}^{(\mathcal{O})}(\mathcal{S}^{(\mathrm{D}^{(\mathcal{O})}(\mathcal{S}))}) \geq 1$, while in fact $\mathrm{D}^{(\mathcal{O})}(\mathcal{S}^{(\mathrm{D}^{(\mathcal{O})}(\mathcal{S}))}) = 0$. This contradiction shows that this last case ($\mathrm{D}^{(\mathcal{O})}(\mathcal{S}) < d - 1$ and $\mathcal{E}^{(\mathrm{D}^{(\mathcal{O})}(\mathcal{S}))}_{m+\mathrm{D}^{(\mathcal{O})}(\mathcal{S})}$ not fulfilling Case 1 of definition of $\mathrm{INV}^{(\mathcal{O})}$) is impossible. We have proved point (3) of the lemma. ∎

## 6. Constants.

**6. Constants.** Let us *fix* a birooted dpda $\mathcal{M}$, a strong relational morphism $\bar{\psi}$, and an initial equation $A_0 = (\Pi_0, S_0^-, S_0^+) \in \mathbb{N} \times \mathsf{DRB}_{1,\lambda_0}\langle\langle\ V\ \rangle\rangle \times \mathsf{DRB}_{1,\lambda_0}\langle\langle\ V\ \rangle\rangle$ in the corresponding set of assertions. This short section is devoted to the definition of some integer *constants*: these integers are constant in the sense that they depend only on this triple: $(\mathcal{M}, \bar{\psi}, A_0)$. The *motivation* of each of these definitions will appear later on, in different places for the different constants. The equations below provide merely an overview of the dependencies between these constants and allow us to check that the definitions are sound (i.e., there is no hidden loop in the dependencies).

$$k_0 = \max\{\nu(v) \mid v \in V\}, \quad k_1 = \max\{2k_0 + 1, 3\}, \tag{6.1}$$

$$K_0 = \max\{\|(E_1, E_2, \ldots, E_n) \odot x\| \mid (E_i)_{1 \leq i \leq n} \text{ is a bijective numbering of some class in } V/\smile,\ x \in X\}. \tag{6.2}$$

$K_0$ serves as an upper-bound on the possible increase of norm under the right-action of a single letter $x \in X$; see Lemma 3.13.

$$D_1 = k_0 \cdot K_0 + |Q| + 2, \quad k_2 = D_1 \cdot k_1 \cdot K_0 + 2 \cdot k_1 \cdot K_0 + K_0. \tag{6.3}$$

$k_1$ is used in the definition of strategy $T_B$ (section 7), $D_1$ appears as an upper-bound on the marked part of series, and $k_2$ is used in Lemma 8.4.

$$k_3 = k_2 + k_1 \cdot K_0, \quad k_4 = (k_3 + 1) \cdot K_0 + k_1. \tag{6.4}$$

$k_3$ appears in in Lemma 8.5, and $k_4$ is used in the definition (8.15) of the d-space $V_0$.

$$\begin{aligned} &K_1 = k_1 \cdot K_0 + 1, \\ &K_2 = k_1^2 \cdot D_1 \cdot K_0 + k_1^2 \cdot K_0 + 2 \cdot k_1 \cdot K_0 + D_1 \cdot k_1 + 2 \cdot k_1 + 4. \end{aligned} \tag{6.5}$$

These constants $K_1, K_2$ appear in Lemma 8.7.

$$K_3 = k_0|Q|, \quad K_4 = D_1. \tag{6.6}$$

These constants $K_3, K_4$ appear in Lemma 8.8.

$$d_0 = \mathrm{Card}(X^{\leq k_4}). \tag{6.7}$$

$d_0$ appears as an upper-bound on the dimension of the d-space $V_0$ defined by equation (8.15) and used in Lemma 8.7. We consider now the integer sequences $(\delta_i, \ell_i, L_i, s_i, S_i, \Sigma_i)_{m \leq i \leq m+n-1}$ defined by the relations (5.19) of section 5.2 where the parameters

$K_0, K_1, \ldots, K_4$ are chosen to be the above constants and $m = 1$, $n = d = d_0$. Equivalently, they are defined by

$$\delta_1 = 0,\ \ell_1 = 0,\ L_1 = K_2,\ s_1 = K_3 \cdot K_2 + K_4,\ S_1 = 0,\ \Sigma_1 = 0, \tag{6.8}$$

$$\begin{cases} \delta_{i+1} = 2 \cdot F(d_0, s_i + \Sigma_i) + 1, \\ \ell_{i+1} = 2 \cdot \delta_{i+1} + 3, \\ L_{i+1} = K_1 \cdot (L_i + \ell_{i+1}) + K_2, \\ s_{i+1} = K_3 \cdot L_{i+1} + K_4, \\ S_{i+1} = s_i + \Sigma_i + K_0 \cdot F(d_0, s_i + \Sigma_i), \\ \Sigma_{i+1} = \Sigma_i + S_{i+1} \end{cases} \tag{6.9}$$

for $1 \leq i \leq d_0 - 1$. The function $F$ is defined in section 5.2 and depends on the pair $(\mathcal{M}, \bar{\psi})$ only.

$$D_2 = \max\{\Sigma_{d_0} + s_{d_0}, \|S_0^-\|, \|S_0^+\|\}. \tag{6.10}$$

$\Sigma_{d_0} + s_{d_0}$ appears in the conclusion of Lemma 5.5 when we take $d = d_0$ in the hypothesis and suppose that $\mathrm{D}^{(\mathcal{O})}(\mathcal{S})$ has its maximal possible value, i.e., $\mathrm{D}^{(\mathcal{O})}(\mathcal{S}) = d_0 - 1$. It is used as an upper-bound on the norm of vectors at the root of the trees $\tau$ analyzed in section 8 (inequality (8.1)).

$$\lambda_2 = \max\{\lambda_0, d_0\}. \tag{6.11}$$

The integer $\lambda_2$ is used as an upper-bound on the length of vectors at the root of the trees $\tau$ analyzed in section 8 (inequality (8.2)).

$$N_0 = 1 + k_3 + D_2. \tag{6.12}$$

$N_0$ appears as a lower-bound for the norm in the definition of a B-stacking sequence (section 8.2, condition (8.5)).

## 7. Strategies for $\mathcal{B}_0$.

Let us define strategies for the particular system $\mathcal{B}_0$.

**7.1. Strategies.** We shall first define auxiliary strategies $T_{cut}, T_\emptyset, T_\varepsilon$; and then for every oracle $\mathcal{O} \in \Omega$ auxiliary strategies $T_A^{(\mathcal{O})}, T_B^{(\mathcal{O})}, T_C^{(\mathcal{O})}$, we define the strategies $T_A, T_B, T_C$ and finally the "compound" strategies $\mathcal{S}_{AB}^{(\mathcal{O})}, \mathcal{S}_{ABC}^{(\mathcal{O})}, \mathcal{S}_{AB}, \mathcal{S}_{ABC}$. Let us fix here some total ordering on $X : x_1 < x_2 < \cdots < x_\alpha$.

- $T_{cut}$:
  $B_1 \cdots B_m \in T_{cut}(A_1 \cdots A_n)$ iff $\exists i \in [1, n-1]$, $\exists S, T$,
  $$A_i = (p_i, S, T),\ A_n = (p_n, S, T),\ p_i < p_n, \text{ and } m = 0.[3]$$
- $T_\emptyset$:
  $B_1 \cdots B_m \in T_\emptyset(A_1 A_2 \cdots A_n)$ iff $\exists S, T$,
  $$A_n = (p, S, T),\ p \geq 0,\ S = T = \emptyset^\lambda, \text{ and } m = 0.$$
- $T_\varepsilon$:
  $B_1 \cdots B_m \in T_\varepsilon(A_1 \cdots A_n)$ iff
  $$A_n = (p, S, T),\ p \geq 0,\ S = T = \varepsilon_i^\lambda \text{ (for some } i \in [1, \lambda]\text{), and } m = 0.$$

[3]That is, $B_1 \cdots B_m = \epsilon$.

Let us consider an oracle $\mathcal{O} \in \Omega$.

- $T_A^{(\mathcal{O})}$:
  $B_1 \cdots B_m \in T_A^{(\mathcal{O})}(A_1 \cdots A_n)$ iff

$$A_n = (p, S, T), \quad | X | \leq m \leq | X |^2,$$

$$B_1 = (p+1, S \odot x_1, T \odot x'_1), \ldots, B_m = (p+1, S \odot x_m, T \odot x'_m),$$

  where $S \not\equiv \varepsilon$, $T \not\equiv \varepsilon$, $\mathcal{O}(S,T) \cap X \times X = \{(x_1, x'_1), \ldots, (x_i, x'_i), \ldots, (x_m, x'_m)\}$.
- $T_B^{(\mathcal{O}),+}$:
  $B_1 \cdots B_m \in T_B^+(A_1 \cdots A_n)$ iff $n \geq k_1 + 1$, $A_{n-k_1} = (\pi, \overline{U}, U')$ (where $\overline{U}$ is unmarked)

$$U' = \sum_{k=1}^{q} E_k \cdot \Phi_k \quad \text{for some } q \in \mathbb{N},\ E_k \in V,$$

  $(E_k)_{1 \leq k \leq q}$ is the bijective numbering of a class in $V/\smile$, $\Phi_k \in \mathsf{DRB}_{1,\lambda}\langle\langle\ V\ \rangle\rangle$, $A_i = (\pi + k_1 + i - n, U_i, U'_i)$ for $n - k_1 \leq i \leq n$, $(U_i)_{n-k_1 \leq i \leq n}$ is a derivation, $(U'_i)_{n-k_1 \leq i \leq n}$ is a "stacking derivation" (see definitions in section 3.3),

$$U'_n = \sum_{k=1}^{q} (E_k \odot u) \cdot \Phi_k \quad \text{for some } u \in X^*,$$

  $m = 1$, $B_1 = (\pi + k_1 - 1, V, V')$, $V = U_n$,

$$V' = \sum_{k=1}^{q} \bar{\rho}_e(E_k \odot u) \cdot (\overline{U} \odot u_k),$$

  where $\forall k \in [1,q]$, $u'_k = \min(\varphi(E_k))$, and if $\mathcal{R} = \mathcal{O}(S,T)$, $\forall k \in [1,q]$, $u_k = \min\{\mathcal{R}^{-1}(u'_k)\}$.
- $T_B^{(\mathcal{O}),-}$:
  $T_B^{(\mathcal{O}),-}$ is defined in the same way as $T_B^{(\mathcal{O}),+}$ by exchanging the left series $(S^-)$ and right series $(S^+)$ in every assertion $(p, S^-, S^+)$.
- $T_C^{(\mathcal{O})}$:
  $B_1 \cdots B_m \in T_C^{(\mathcal{O})}(A_1 \cdots A_n)$ iff there exists $d \in [1, d_0]$, $D \in [0, d-1]$, $\lambda \in \mathbb{N} - \{0\}$, $S_1, S_2, \ldots, S_d \in \mathsf{DRB}_{1,\lambda}\langle\langle\ V\ \rangle\rangle - \{\emptyset^\lambda\}$, $1 \leq \kappa_1 < \kappa_2 < \cdots < \kappa_{D+1} = n$, such that
  (C1) every equation $\mathcal{E}_i = A_{\kappa_i} = (p_{\kappa_i} S^-_{p_{\kappa_i}}, S^+_{p_{\kappa_i}})$ is a weighted equation over $S_1, S_2, \ldots, S_d$, with $p_{\kappa_i} \geq 1$;
  (C2) $\mathrm{D}^{(\mathcal{O})}(\mathcal{S}) = D$ (where $\mathcal{S} = (\mathcal{E}_i)_{1 \leq i \leq D+1}$);
  (C3) $\mathrm{INV}^{(\mathcal{O})}(\mathcal{S}) \neq \perp$, $|||\ \mathrm{INV}^{(\mathcal{O})}(\mathcal{S})\ ||| \leq \Sigma_{d_0} + s_{d_0}$;
  (C4) $m = 1$ and $B_1 = \rho_e(\mathrm{INV}^{(\mathcal{O})}(\mathcal{S}))$ (where $\rho_e$ is the obvious extension of $\rho_e$ to weighted pairs of deterministic row-vectors; in other words the result of $T_C^{(\mathcal{O})}$ is $\mathrm{INV}^{(\mathcal{O})}(\mathcal{S})$, where the marks have been removed).

We then set, for every $W \in \mathcal{A}^+$,

$$T_A(W) = \bigcup_{\mathcal{O} \in \Omega} T_A^{(\mathcal{O})}(W),$$

$$T_B^+(W) = \bigcup_{\mathcal{O} \in \Omega} T_B^{(\mathcal{O}),+}(W), \quad T_B^-(W) = \bigcup_{\mathcal{O} \in \Omega} T_B^{(\mathcal{O}),-}(W),$$

$$T_C(W) = \bigcup_{\mathcal{O} \in \Omega} T_C^{(\mathcal{O})}(W).$$

LEMMA 7.1. *$T_{cut}, T_\emptyset, T_\varepsilon, T_A$ are $\mathcal{B}_0$-strategies.*

*Proof.*

$T_{cut}$: (S1) is true by rule (R0). (S2) is trivially true.

$T_\emptyset$: (S1) is true by rule (R$'$3). (S2) is trivially true.

$T_\varepsilon$: (S1) is true by rule (R$'$3). (S2) is trivially true.

$T_A$: By rule (R4), $\{B_j \mid 1 \leq j \leq m\} \mid\!\!\vdash\!\!-_4 A_n$, which proves (S1). Suppose $H(A_n) = \infty$, i.e., $S \sim T$. Then, $\forall j \in [1, m]$, $S \odot x_j \sim T \odot x'_j$, so that $\min\{H(B_j) \mid 1 \leq j \leq m\} = \infty$. (S2) is proved. □

LEMMA 7.2. *$T_B^+, T_B^-$ are $\mathcal{B}_0$-strategies.*

*Proof.* Let us show that $T_B^+$ is a $\mathcal{B}_0$-strategy.

We use the notation of the definition of $T_B^{(\mathcal{O}),+}$. Let $\mathcal{H} = \{(\pi, \overline{U}, U'), (\pi + k_1 - 1, V, V')\}$. Let us show that

$$\mathcal{H} \overset{\langle * \rangle}{\mid\!\!\vdash\!\!-}_{\mathcal{B}_0} (\pi + k_1 - 1, U_n, U'_n). \tag{7.1}$$

Using rule (R5) we obtain, $\forall k \in [1, q]$,

$$\begin{aligned}\{(\pi, \overline{U}, U')\} = \left\{\left(\pi, \overline{U}, \sum_{j=1}^{q} E_j \cdot \Phi_j\right)\right\} &\overset{\langle * \rangle}{\mid\!\!\vdash\!\!-}_{R5} (\pi + 2 \cdot \mid u_k \mid, \overline{U} \odot u_k, U' \odot u'_k) \\ &\overset{\langle * \rangle}{\mid\!\!\vdash\!\!-}_{R0} (\pi + 2 \cdot k_0, \overline{U} \odot u_k, U' \odot u'_k) \\ &= \quad (\pi + 2 \cdot k_0, \overline{U} \odot u_k, \Phi_k).\end{aligned} \tag{7.2}$$

Using rule (R$'$3),

$$\emptyset \mid\!\!\vdash\!\!-_{R'3} (0, (\rho_e(E_1 \odot u), \ldots, \rho_e(E_q \odot u)), (E_1, \ldots, E_q)). \tag{7.3}$$

Using (7.3), (7.2) and rules (R3), (R7), (R8), we obtain

$$\begin{aligned}\{(\pi, \overline{U}, U')\} &\overset{\langle * \rangle}{\mid\!\!\vdash\!\!-}_{\mathcal{B}_0} \left(\pi + 2k_0, \sum_{k=1}^{q} (E_k \odot u) \cdot \Phi_k, \sum_{k=1}^{q} \rho_e(E_k \odot u) \cdot (\bar{U} \odot u_k)\right) \\ = \quad \{(\pi, \overline{U}, U')\} &\overset{\langle * \rangle}{\mid\!\!\vdash\!\!-} (\pi + 2k_0, U'_n, V').\end{aligned} \tag{7.4}$$

Let us recall that $U_n = V$. Hence, by (R0), (R1), (R2)

$$\{(\pi + k_1 - 1, V, V'), (\pi + 2k_0, U'_n, V')\} \overset{\langle * \rangle}{\mid\!\!\vdash\!\!-}_{\mathcal{C}} (\pi + k_1 - 1, U_n, U'_n). \tag{7.5}$$

By (7.4) and (7.5), (7.1) is proved. Now, using (7.1) and rule (R0), we obtain

$$\mathcal{H} \overset{\langle * \rangle}{||\!\!-\!\!-}_{\mathcal{B}_0} (\pi + k_1 - 1, U_n, U'_n) \vdash\!\!-_{R0} (\pi + k_1, U_n, U'_n), \tag{7.6}$$

i.e., $T_B^+$ fulfills (S1).

Let us suppose now that $\forall i \in [n - k_1, n]$, $U_i \sim U'_i$. Then, by (7.4), $U'_n \sim V'$ and by the hypothesis, $V = U_n \sim U'_n$. Hence $V \sim V'$. This shows that $T_B^+$ fulfills (S2).

An analogous proof can obviously be written for $T_B^-$. ◻

LEMMA 7.3. *Let $(p, S, S')$ be a weighted equation, i.e., $p \in \mathbb{N}$, $\lambda \in \mathbb{N} - \{0\}$, $S, S' \in \mathsf{DRB}_{1,\lambda}\langle\langle\ V\ \rangle\rangle$. Then*

$$\{(p, S, S')\} \overset{\langle * \rangle}{||\!\!-\!\!-}_{\mathcal{C}} \{(p, \rho_e(S), \rho_e(S'))\},$$

$$\{(p, \rho_e(S), \rho_e(S'))\} \overset{\langle * \rangle}{||\!\!-\!\!-}_{\mathcal{C}} \{(p, S, S')\}.$$

*Proof.* The proof follows easily from (R1), (R2), (R′3). ◻

LEMMA 7.4. *For every $\mathcal{O} \in \Omega$, $T_C^{\mathcal{O}}$ is a $\mathcal{B}_0$-strategy.*

*Proof.* By Lemma 5.3, point (1), combined with Lemma 7.3, (S1) is proved. By Lemma 5.3, point (2), combined with Lemma 7.3, (S2) is proved. ◻

Let us define the strategy $\mathcal{S}_{ABC}$ by the following: for every $W = A_1 A_2 \cdots A_n$,

(0) if $T_{cut}(W) \neq \emptyset$, then $\mathcal{S}_{ABC}(W) = T_{cut}(W)$;
(1) else if $T_\emptyset(W) \neq \emptyset$, then $\mathcal{S}_{ABC}(W) = T_\emptyset(W)$;
(2) else if $T_\varepsilon(W) \neq \emptyset$, then $\mathcal{S}_{ABC}(W) = T_\varepsilon(W)$;
(3) else if $T_B^+(W) \cup T_B^-(W) \neq \emptyset$, then $\mathcal{S}_{ABC}(W) = T_B^+(W) \cup T_B^-(W) \cup T_C(W)$;
(4) else $\mathcal{S}_{ABC}(W) = T_A(W) \cup T_C(W)$.

The strategy $\mathcal{S}_{AB}$ is obtained from $\mathcal{S}_{ABC}$ by removing the occurrence of $T_C$ in cases (3) and (4).

**7.2. Global strategy.** Let us define a global strategy $\hat{\mathcal{S}}_{ABC}$ w.r.t. the strategy $\mathcal{S}_{ABC}$. Let us fix (until the end of this article) a total well-ordering $\sqsubseteq$ over the set of oracles $\Omega$. We need now three technical definitions.

DEFINITION 7.5. *Let $P \in \mathcal{P}_f(\mathcal{A})$, $\mathcal{O} \in \Omega$, and $\bar{\pi} \in \mathbb{N} \cup \{\infty\}$. $\mathcal{O}$ is* $\bar{\pi}$-consistent *with $P$ iff, for every $(\pi, S, S') \in \mathrm{Cong}(P)$ and every $n \in \mathbb{N}$, if*

$$\pi + n - 1 < \bar{\pi},$$

*then the binary relation $\mathcal{R}_n = \mathcal{O}(S, S') \cap X^{\leq n} \times X^{\leq n}$ fulfills*

$$[\pi, S, S', \mathcal{R}_n] \subseteq \mathrm{Cong}(P).$$

We use the notation

$$\Omega(\bar{\pi}, P) = \{\mathcal{O} \in \Omega \mid \mathcal{O} \text{ is } \bar{\pi}\text{-consistent with } P\}.$$

DEFINITION 7.6. *Let $P$ be a finite subset of $\mathcal{A}$, and let $\bar{\pi} \in \mathbb{N} \cup \{\infty\}$. $P$ is said to be* $\bar{\pi}$-consistent *iff there exists some oracle $\mathcal{O} \in \Omega$, which is $\bar{\pi}$-consistent with $P$.*

For every proof-tree $t \in \mathcal{T}(\mathcal{S}_{ABC})$, we denote by $\bar{\Pi}(t)$ the integer

(7.7)
$$\bar{\Pi}(t) = \min\{\pi \in \mathbb{N} \mid \exists x \in \mathrm{dom}(t),\ x \text{ is not closed for } \mathcal{S}_{ABC},\ \exists S, S',\ t(x) = (\pi, S, S')\}.$$

(We admit here that $\min(\emptyset) = \infty$.)

DEFINITION 7.7. *Let $t$ be a finite proof-tree for the strategy $\mathcal{S}_{ABC}$, $t \in \mathcal{T}(\mathcal{S}_{ABC})$. $t$ is* consistent *iff* $\mathrm{im}(t)$ *is* $\bar{\Pi}(t)$*-consistent.*

Let us consider some tree $t \in \mathcal{T}(\mathcal{S}_{ABC})$ which is consistent and not closed. Let $\bar{\pi} = \bar{\Pi}(t)$, and let $x$ be the smallest unclosed node of weight $\bar{\pi}$. Let

$$W = A_1 \cdots A_n \tag{7.8}$$

be the word labeling the path from the root to $x$ in $t$. (One can notice that, as $x$ is not closed, $T_{cut}(W) \cup T_{\emptyset}(W) \cup T_{\varepsilon}(W) = \emptyset$.) We define a tree of height one, $\hat{\Delta}(t)$, as follows:

(0) If $\exists \mathcal{O} \in \Omega(\bar{\pi}, \mathrm{im}(t))$, $T_C^{(\mathcal{O})}(W) \neq \emptyset$, then

$$\mathcal{O}_0 = \min\{\mathcal{O} \in \Omega(\bar{\pi}, \mathrm{im}(t)), T_C^{(\mathcal{O})}(W) \neq \emptyset\}, \quad \hat{\Delta}(t) = A_n(T_C^{(\mathcal{O}_0)}(W));$$

(1) else if $T_B^{+}(W) \neq \emptyset$, then

$$\mathcal{O}_0 = \min(\Omega(\bar{\pi}, \mathrm{im}(t))), \quad \hat{\Delta}(t) = A_n(T_B^{(\mathcal{O}_0),+}(W));$$

(2) else if $T_B^{-}(W) \neq \emptyset$, then

$$\mathcal{O}_0 = \min(\Omega(\bar{\pi}, \mathrm{im}(t))), \quad \hat{\Delta}(t) = A_n(T_B^{(\mathcal{O}_0),-}(W));$$

(3) else

$$\mathcal{O}_0 = \min(\Omega(\bar{\pi}, \mathrm{im}(t))), \quad \hat{\Delta}(t) = A_n(T_A^{(\mathcal{O}_0)}(W)).$$

(In the above definition, by $A(W')$, where $A \in \mathcal{A}$, $W' \in \mathcal{A}^+$, we mean the tree of height one with root labeled by $A$ and whose sequence of leaves is the word $W'$.)

$$\hat{\mathcal{S}}_{ABC}(t) = t[\hat{\Delta}(t)/x], \tag{7.9}$$

i.e., $\hat{\mathcal{S}}_{ABC}(t)$ is obtained from $t$ by substituting $\hat{\Delta}(t)$ at the leaf $x$.

LEMMA 7.8. *For every $t \in \mathcal{T}(\mathcal{S}_{ABC})$, if $t$ is consistent, then $\hat{\Delta}(t)$ is defined.*

*Proof.* By the definition of consistency the oracle $\mathcal{O}_0$ is always defined (that is, $\Omega(\bar{\pi}, \mathrm{im}(t)) \neq \emptyset$), and for the word $W$ defined above, $T_{\varepsilon}(W) = \emptyset \Rightarrow \forall \mathcal{O} \in \Omega$, $T_A^{(\mathcal{O})}(W) \neq \emptyset$; hence one of cases (0)–(3) must occur. $\square$

If $t$ is not consistent or is closed, then we define

$$\hat{\mathcal{S}}_{ABC}(t) = t. \tag{7.10}$$

LEMMA 7.9. *$\hat{\mathcal{S}}_{ABC}$ is a global strategy for $\mathcal{S}_{ABC}$.*

*Sketch of proof.* By Lemma 7.8 $\hat{\mathcal{S}}_{ABC}$ is defined on every $t \in \mathcal{T}(\mathcal{S}_{ABC})$. It suffices to check that, in every case, the word constituted by the leaves of $\hat{\Delta}(t)$ belongs to $\mathcal{S}_{ABC}(W)$ (where $W$ is the word considered in (7.8)). $\square$

## 8. Tree analysis.

**8. Tree analysis.** This section is devoted to the analysis of the proof-trees $\tau$ produced by the strategy $\mathcal{S}_{AB}$ defined in section 7. The main results are Lemmas 8.9 and 8.10, whose combination asserts that if some branch of $\tau$ is infinite, then there exists some finite prefix on which $T_C$ has a nonempty value. This key technical result will ensure termination of the global strategy $\hat{\mathcal{S}}_{ABC}$ (see section 9).

We fix throughout this section a tree $\tau \in \mathcal{T}(\mathcal{S}_{AB}, (\pi_0, U_0^-, U_0^+))$ (i.e., $\tau$ is a proof-tree associated to the assertion $(\pi_0, U_0^-, U_0^+))$ by the strategy $\mathcal{S}_{AB}$). We suppose that

$$\|U_0^-\| \leq D_2, \quad \|U_0^+\| \leq D_2, \quad U_0^-, U_0^+ \text{ are both unmarked,} \tag{8.1}$$

$$U_0^-, U_0^+ \in \mathsf{DRB}_{1,\lambda}\langle\langle\, V\, \rangle\rangle \text{ with } \lambda \leq \lambda_2, \tag{8.2}$$

$$U_0^- \equiv U_0^+. \tag{8.3}$$

We recall that, formally, $\tau$ is a map $\mathrm{dom}(\tau) \to \mathbb{N} \times \mathsf{DRB}_{1,\lambda}\langle\langle\, V\, \rangle\rangle \times \mathsf{DRB}_{1,\lambda}\langle\langle\, V\, \rangle\rangle$ such that $\mathrm{dom}(\tau) \subseteq \{1, \ldots, |X|^2\}^*$ is closed under prefix and under "left-brother" (i.e., $w \cdot (i+1) \in \mathrm{dom}(\tau) \Rightarrow w \cdot i \in \mathrm{dom}(\tau)$). We denote by $pr_{2,3} : \mathbb{N} \times \mathsf{DRB}_{1,\lambda}\langle\langle\, V\, \rangle\rangle \times \mathsf{DRB}_{1,\lambda}\langle\langle\, V\, \rangle\rangle \to \mathsf{DRB}_{1,\lambda}\langle\langle\, V\, \rangle\rangle \times \mathsf{DRB}_{1,\lambda}\langle\langle\, V\, \rangle\rangle$ the projection $(\pi, U, U') \mapsto (U, U')$. By $\tau_s$ we denote the tree obtained from $\tau$ by forgetting the weights $\tau_s = \tau \circ pr_{2,3}$.

**8.1. Depth and weight.** In this paragraph we check that the *weight* and the *depth* of a given node are closely related. Let us say that the strategy $T$ "occurs at" node $x$ iff

$$\tau(x) \in T(\tau(x[0]) \cdot \tau(x[1]) \cdots \tau(x[|x| - 1]));$$

i.e., the label of $x$ belongs to the image of the path from $\epsilon$ (included) to $x$ (excluded) by the strategy $T$.

LEMMA 8.1. *Let $\alpha \in \{-, +\}$, $A_1, \ldots, A_n \in \mathcal{A}$ such that $T_B^\alpha(A_1 \cdots A_n) \neq \emptyset$. Then, $\forall i \in [n - k_1 + 1, n]$, $A_i \notin T_B(A_1 \cdots A_{i-1})$.*

In other words, if $T_B$ occurs at node $x$ of $\tau$, it cannot occur at any of its $k_1$ above immediate ancestors.

*Proof.* Suppose that $\exists i \in [n - k_1 + 1, n]$, $A_i \in T_B(A_1 \cdots A_{i-1})$. Hence $\pi_i = \pi_{i-1} - 1 < \pi_{n-k_1} + i$, contradicting one of the hypotheses under which $T_B(A_1 \cdots A_n)$ is not empty. □

Lemma 8.1 ensures that, in every branch $(x_i)_{i \in I}$ and for every interval $[n + 1, n + 4] \subseteq I$, at most one integer $j$ is such that $T_B$ occurs at $j$.

LEMMA 8.2. *Let $\tau$ be a proof-tree associated to the strategy $\mathcal{S}_{AB}$. Let $x, x' \in \mathrm{dom}(\tau)$, $x \preceq x'$. Then $\mid W(x') - W(x) \mid \leq |x'| - |x| \leq 2 \cdot (W(x') - W(x)) + 3$.*

(We recall the *depth* of a node $x$ is just its length $|x|$.) We denote by $W(x)$ the weight of $x$ which we define as the first component of $\tau(x)$, i.e., the weight of the equation labeling $x$.

*Proof.* Let $x, x'$ be such that $|x'| = |x| + 1$. Then $W(x') - W(x) \in \{-1, +1\}$, and hence the inequality $\mid W(x') - W(x) \mid \leq |x'| - |x|$ is fulfilled by such nodes. The general case follows by induction on $(|x'| - |x|)$.

Let us prove now the other inequality. We distinguish two cases.

*Case* 1. $|x'| - |x| \leq 3$. Then $|x'| - |x| \leq 2 \cdot (W(x') - W(x)) + 3$ (because there is at most one $T_B$ step in a sequence of length $\leq 3$).

*Case* 2. $|x'| - |x| \geq 4$. Let $x = x_0, x_1, \ldots, x_q, x'$ be the sequence of nodes such that $|x'| - |x| = 4 \cdot q + r$, $0 \leq r < 4$, and $\forall i \in [0, q - 1]$, $|x_{i+1}| - |x_i| = 4$.

By Lemma 8.1, in every set $\{y \in \mathrm{dom}(\tau) \mid x_i \prec y \preceq x_{i+1}\}$ at most one node $z$ is such that $T_B$ occurs at $z$. Hence $W(x_{i+1}) - W(x_i) \geq 2$.

It follows that

$$\begin{aligned}
|x'| - |x| &= \sum_{i=0}^{q-1}[|x_{i+1}| - |x_i|] + |x'| - |x_q| \\
&\leq \sum_{i=0}^{q-1} 2(W(x_{i+1} - W(x_i)) + |x'| - |x_q| \\
&\leq 2(W(x_q) - W(x)) + 2(W(x') - W(x_q)) + 3 \ \text{(by the first case)} \\
&\leq 2(W(x') - W(x)) + 3. \qquad \square
\end{aligned}$$

Let us recall the values of some constants (defined in section 6):

$$k_0 = \max\{\nu(v) \mid v \in V\}, \quad k_1 = \max\{2k_0 + 1, 3\}, \quad D_1 = k_0 \cdot K_0 + |Q| + 2,$$

$$k_2 = D_1 \cdot k_1 \cdot K_0 + 2 \cdot k_1 \cdot K_0 + K_0, \quad k_3 = k_2 + k_1 \cdot K_0,$$

$$k_4 = (k_3 + 1) \cdot K_0 + k_1, \quad d_0 = \mathrm{Card}(X^{\leq k_4}), \quad N_0 = 1 + k_3 + D_2.$$

**8.2. B-stacking sequences.** We establish here that every infinite branch must contain an infinite suffix (a "B-stacking sequence") where at least $d_0$ labels $(U, U')$ belong to the same d-space $V_0$ of dimension $\leq d_0$ with coordinates not greater than $s_{d_0}$ (over some fixed generating family of cardinality $\leq d_0$).

Let $\sigma = (x_i)_{i \in I}$ be a path in $\tau$, where $I = [i_0, \infty[$ and let $(x_i)_{i \geq 0}$ be the unique branch of $\tau$ containing $\sigma$. Let us note $\tau(x_i) = (\pi_i, U_i^-, U_i^+)$.

We call $\sigma$ a *B-stacking sequence* iff there exists some $\alpha_0 \in \{-, +\}$ such that

$$T_B^{\alpha_0} \text{ occurs at } x_{i_0+k_1+1}, \tag{8.4}$$

and, for every $i \in I, \alpha \in \{-, +\}$, if $T_B^{\alpha}$ occurs at $x_{i+k_1+1}$, then

$$\|U_i^{-\alpha}\| \geq \|U_{i_0}^{-\alpha_0}\| \geq N_0. \tag{8.5}$$

From now on and until Lemma 8.10, we fix a B-stacking sequence $\sigma = (x_i)_{i \in I}$ and denote by $S_0$ the series $U_{i_0}^{-\alpha_0}$.

LEMMA 8.3. *There exists some word $u_0 \in X^*$ and some sign $\alpha_0' \in \{-, +\}$ such that $S_0 = U_0^{\alpha_0'} \odot u_0$.*

*Proof.* One can prove by induction on $i \in \mathbb{N}$ that, for every $\alpha \in \{-, +\}$, $U_i^{\alpha}$ has one of the two following forms:

(1) $U_i^{\alpha} = U_0^{\alpha'} \odot u$ for some $\alpha' \in \{-, +\}$, $|u| \leq i$;

(2) $U_i^{\alpha} = \sum_{k=1}^{q} \beta_k \cdot (U_0^{\alpha'} \odot uu_k)$

for some deterministic rational vector $\beta$, $\alpha' \in \{-, +\}$, $|u \cdot u_k| \leq i$, $|u_k| \leq k_0$. $\square$

LEMMA 8.4. *Suppose that $i_0 \leq j < i$, no $T_B$ occurs in $[j + 1, i]$, $U_j^{-\alpha}$ is $D_1$-marked, and $U_j^{\alpha}$ is unmarked. Then, for every $j' \in [j, i]$, $\|U_{j'}^{\alpha}\| \geq \|U_i^{\alpha}\| - k_2$.*

*Proof.* Let $i, j$ fulfill the hypothesis of the lemma.

(1) Let us first treat the case where $j' = j$. If $(i - j) \leq (D_1 + 1)k_1$, then, by Lemma 3.13,

$$\|U_i^{\alpha}\| \leq \|U_j^{\alpha}\| + (D_1 + 1) \cdot k_1 \cdot K_0 \leq k_2;$$

hence the lemma is true.

Let us suppose now that $(i - j) \geq (D_1 + 1)k_1 + 1$. We can then define the integers $j < i_1 < i_2 < i$ by

$$i_1 = j + D_1 \cdot k_1, \ i_2 = i - k_1 - 1.$$

By Lemma 3.13 we know that

$$\|U^{\alpha}_{i_1}\| \leq \|U^{\alpha}_{j}\| + D_1 \cdot k_1 \cdot K_0 \quad \text{and} \quad \|U^{\alpha}_{i}\| \leq \|U^{\alpha}_{i_2}\| + (k_1+1) \cdot K_0. \tag{8.6}$$

If there were some stacking subderivation of length $k_1$ in $U^{-\alpha}_{j} \to U^{-\alpha}_{i_1}$, as all the $U^{\alpha}_{k}$ (for $k \in [j,i]$) are unmarked, $T_B$ would occur at some integer in $[j+k_1+1, i_1+1]$, which is untrue. Hence there is no such stacking subderivation, and by Lemma 3.32 $U^{-\alpha}_{i_1}$ is unmarked.

If there were some stacking subderivation of length $k_1$ in $U^{\alpha}_{i_1} \to U^{\alpha}_{i_2}$, as all the $U^{-\alpha}_{k}$ (for $k \in [i_1,i]$) are unmarked, $T_B$ would occur at some integer in $[i_1+k_1+1, i]$, which is untrue. Hence there is no such stacking subderivation, and by Lemma 3.31

$$\|U^{\alpha}_{i_2}\| \leq \|U^{\alpha}_{i_1}\| + k_1 \cdot K_0. \tag{8.7}$$

Adding inequalities (8.6), (8.7) we obtain

$$\|U^{\alpha}_{i}\| \leq \|U^{\alpha}_{j}\| + (D_1 \cdot k_1 + 2 \cdot k_1 + 1) \cdot K_0 = \|U^{\alpha}_{j}\| + k_2,$$

which was to be proved.

(2) Let us suppose now that $j \leq j' \leq i$. If $(i-j) \leq (D_1+1)k_1$, the same inequality is true for $i-j'$ and the conclusion is true for $j'$.

Otherwise, if $j' \leq i_1$, then (8.6), (8.7) are still true for $j'$ instead of $j$, and hence the conclusion too.

Otherwise, by the arguments of part (1), $U^{-\alpha}_{j'}, U^{\alpha}_{j'}$ are both unmarked. Therefore the hypotheses of part (1) are met by $(j', i)$ instead of $(j,i)$, and hence the conclusion is met too. (We illustrate our argument in Figure 4.) □

LEMMA 8.5. *Let $i \in I$, $\alpha \in \{-,+\}$ such that $T^{\alpha}_{B}$ occurs at $i+k_1+1$. Then there exists $u \in X^*$, $|u| \leq (i-i_0)$, $U^{-\alpha}_{i} = S_0 \odot u$, and, for every prefix $w \preceq u$,*

$$\|S_0 \odot w\| \geq \|S_0\| - k_3.$$

*Proof.* We prove the lemma by induction on $i \in [i_0, \infty[$.

**Basis:** $i = i_0$.

Choosing $u = \epsilon$, the lemma is true.

**Induction step:** $i_0 \leq i' < i$, $T^{\alpha'}_{B}$ occurs at $i'+k_1+1$, $T^{\alpha}_{B}$ occurs at $i+k_1+1$, and $T_B$ does not occur in $[i'+k_1+2, i+k_1]$.

By the induction hypothesis, there exists some $u' \in X^*$, $|u'| \leq (i'-i_0)$, fulfilling

$$U^{-\alpha'}_{i'} = S_0 \odot u', \tag{8.8}$$

$$\forall w' \preceq u', \quad \|S_0 \odot w'\| \geq \|S_0\| - k_3. \tag{8.9}$$

Let us define $j = i'+k_1+1$.

Let $\bar{u} \in X^*$ be the word such that

$$U^{-\alpha}_{j} \xrightarrow{\bar{u}} U^{-\alpha}_{i} \tag{8.10}$$

is the derivation described by the $-\alpha$ component of the path from $x_j$ to $x_i$.

*Case* 1. $\alpha' = \alpha$.

$$U^{-\alpha}_{j} = U^{-\alpha'}_{i'} \odot u_1$$

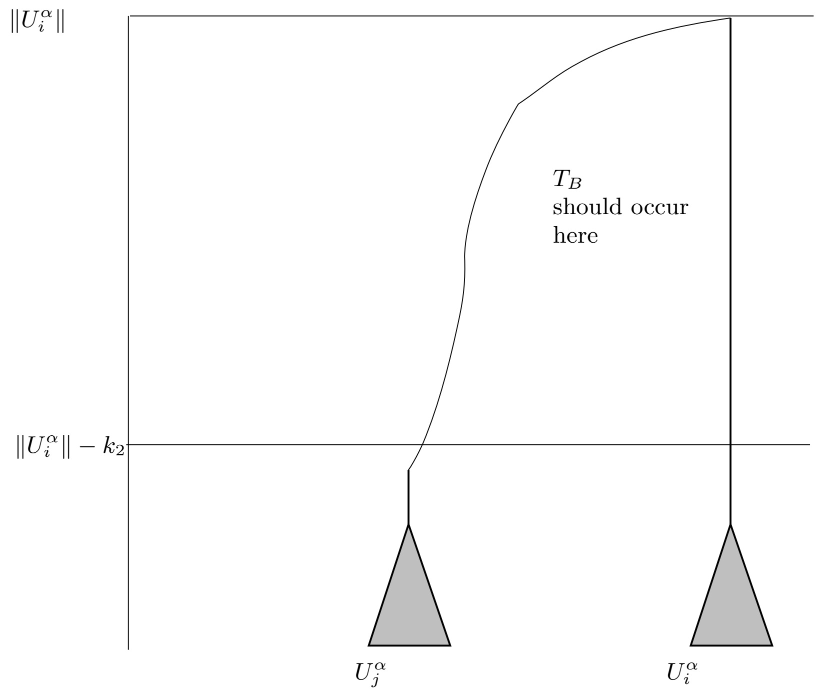


FIG. 4. $\|U_j^\alpha\|$ *too small is impossible.*

for some $u_1 \in X^*$, $|u_1| = k_1$, and $U_j^\alpha$ is $D_1$-marked. Let us choose $u = u' \cdot u_1 \cdot \bar{u}$. Hence

$$U_i^{-\alpha} = S_0 \odot u. \tag{8.11}$$

Let us consider some prefix $w$ of $u$.

*Subcase* 1. $w \preceq u'$. By (8.9) we know that $\|S_0 \odot w\| \geq \|S_0\| - k_3$.

*Subcase* 2. $w = u' \cdot u_1 \cdot u''$ for some $u'' \preceq \bar{u}$. By Lemma 8.4 we know that $\|S_0 \cdot w\| \geq \|U_i^\alpha\| - k_2$, and by the definition of a B-stacking sequence we also know that $\|U_i^\alpha\| \geq \|S_0\|$. Hence

$$\|S_0 \odot w\| \geq \|S_0\| - k_2.$$

*Subcase* 3. $w = u' \cdot u_1'$, where $u_1'$ is a prefix of $u_1$. Then, by Lemma 3.13 and the above inequality, we get

$$\|S_0 \odot w\| \geq \|S_0 \odot u'u_1\| - k_1 \cdot K_0 \geq \|S_0\| - k_3.$$

*Case* 2. $-\alpha' = \alpha$.

$$U_j^{-\alpha} = \sum_{k=1}^{q} \beta_k \cdot (U_{i'}^\alpha \odot u_k),$$

where $\beta$ is a polynomial which is fully marked and every $|u_k| \leq k_0$.

By Lemma 3.15 either $U_i^{-\alpha} = \sum_{k=1}^{q} (\beta_k \odot \bar{u}) \cdot (U_{i'}^\alpha \odot u_k)$ or there exists a decomposition

$$\bar{u} = \bar{u}_1 \cdot \bar{u}_2 \tag{8.12}$$

and an integer $k \in [1, q]$ such that

$$U_i^{-\alpha} = U_{i'}^{\alpha} \odot u_k \bar{u}_2. \tag{8.13}$$

But, as $U_i^{-\alpha}$ is unmarked (by definition of $T_B^{\alpha}$), the first formula is impossible unless $\beta \odot \bar{u}$ is unitary or null. Hence (8.12), (8.13) is the only possibility.

Let us choose $u = u' \cdot u_k \cdot \bar{u}_2$. It is clear from (8.13) that $U_i^{-\alpha} = S_0 \odot u$.

Let us consider some prefix $w$ of $u$.

*Subcase* 1. $w \preceq u'$. We use the same arguments as in Case 1, Subcase 1.

*Subcase* 2. $w = u' \cdot u_k \cdot u''$ for some $u'' \preceq \bar{u}_2$. By Lemma 8.4 applied to the interval $[j + |\bar{u}_1| + 1, i]$, we can conclude that

$$\|S_0 \odot w\| \geq \|S_0\| - k_3.$$

*Subcase* 3. $w = u' \cdot u'_k$, where $u'_k$ is a prefix of $u_k$. We use the same arguments as in Case 1, Subcase 3. $\square$

Let us now define the following set of vectors and d-spaces:

$$\mathcal{G}_0 = \{S_0 \odot u \mid u \in X^*, |u| \leq k_4\}, \tag{8.14}$$

$$V_0 = \mathsf{V}(\mathcal{G}_0). \tag{8.15}$$

LEMMA 8.6. *Let $i \geq i_0$ such that $T_B$ occurs at $i$. Then $U_i^-, U_i^+ \in V_0$.*

*Proof.* Let us suppose that $T_B^{\alpha}$ occurs at $i$. By Lemma 8.5, $U_{i-k_1-1}^{-\alpha} = S_0 \odot u$ and, for every prefix $w \preceq u$,

$$\|S_0 \odot w\| \geq \|S_0\| - k_3.$$

By Lemma 3.14, $\exists u_1, u_2 \in X^*$, $v_1 \in V^*$, $E_1, \ldots, E_k \in V$, $E_1 \smile E_2 \smile \cdots \smile E_k$, $\Phi \in \mathsf{DRB}_{q,\lambda}\langle\langle\ V\ \rangle\rangle$, such that $u = u_1 \cdot u_2$,

$$S_0 \odot u_1 = S_0 \bullet v_1 = \sum_{k=1}^{q} E_k \cdot \Phi_k, \tag{8.16}$$

$$S_0 \odot u = \sum_{k=1}^{q} (E_k \odot u_2) \cdot \Phi_k. \tag{8.17}$$

Without loss of generality, we can suppose that $v_1$ is a minimal word realizing the equality (8.16). Let us notice that, as $G$ is a reduced grammar, for every $v \preceq v_1$, there exists some $\bar{v} \in X^*$, such that $S_0 \bullet v = S_0 \odot \bar{v}$. Hence, for every $v \preceq v_1$,

$$S_0 \bullet v = U_0^{\alpha'_0} \odot u_0 \cdot \bar{v} \quad \text{and} \quad \|U_0^{\alpha'_0} \odot u_0 \cdot \bar{v}\| \geq \|S_0 \odot u_1\| > D_2 = \|U_0^{\alpha'_0}\|.$$

By Lemma 3.29, all the vectors $S_0 \bullet v$ for $v \preceq v_1$ are loop-free. It follows that, for every $v \preceq v' \preceq v_1$,

$$v \prec v' \Rightarrow \|S_0 \bullet v\| > \|S_0 \bullet v'\|,$$

and hence

$$|v_1| \leq \|S_0\| - \|S_0 \bullet v_1\| \leq k_3.$$

The formula (8.17) can be rewritten

$$U^{-\alpha}_{i-k_1-1} = \sum_{k=1}^{q}(E_k \odot u_2) \cdot (S_0 \bullet v_1 E_k) = \sum_{k=1}^{q}(E_k \odot u_2) \cdot (S_0 \odot \bar{u}_k),$$

where $\bar{u}_k \in X^*$, $|\bar{u}_k| \leq (k_3 + 1) \cdot K_0$.

Using Lemmas 3.15 and 3.11 we can deduce from the above form of $U^{-\alpha}_{i-k_1-1}$ that

$$U^{\alpha}_{i} \in \mathsf{V}(\{S_0 \odot w \mid w \in X^*, |w| \leq (k_3 + 1) \cdot K_0 + k_0\}),$$
$$U^{-\alpha}_{i} \in \mathsf{V}(\{S_0 \odot w \mid w \in X^*, |w| \leq (k_3 + 1) \cdot K_0 + k_1\}),$$

and hence that both $U^{-\alpha}_i, U^{\alpha}_i$ belong to $V_0$. □

We recall that

$$K_1 = k_1 \cdot K_0 + 1, \quad K_2 = k_1^2 \cdot D_1 \cdot K_0 + k_1^2 \cdot K_0 + 2 \cdot k_1 \cdot K_0 + D_1 \cdot k_1 + 2 \cdot k_1 + 4.$$

LEMMA 8.7. *For every $L \geq 0$ there exists $i \in [i_0 + L, i_0 + K_1 \cdot L + K_2]$ such that $U^-_i, U^+_i \in V_0$.*

*Proof.* Let us establish that

(8.18)
$$\exists i \in [i_0 + L, i_0 + K_1 \cdot L + K_2 - k_1 - 1], \exists \alpha \in \{-, +\}, \quad T^{\alpha}_{B} \text{ occurs at } i + k_1 + 1.$$

Let $L \geq 0$ and let $i' \geq i_0$ be the greatest integer in $[i_0, i_0 + L]$ such that $T_B$ occurs at $i' + k_1 + 1$. Let $j = i' + k_1 + 1$. We then have

$$U^{\alpha'}_{j} = \sum_{k=1}^{q} \beta_k \cdot (U^{-\alpha'}_{i'} \odot u_k),$$

where $\|\beta\| \leq D_1$ and $U^{-\alpha'}_{j}$ is unmarked.

*Case* 1. *There exists $i \in [j, j + k_1 \cdot D_1]$, such that $T_B$ occurs at $i + k_1 + 1$.* In this case the small constants $K_1 = 0$, $K_2 = k_1 \cdot D_1 + k_1 + 1$ would be sufficient to satisfy (8.18). A fortiori the given constants satisfy (8.18).

*Case* 2. *There exists no $i \in [j, j + k_1 \cdot D_1]$, such that $T_B$ occurs at $i + k_1 + 1$.* Then there is no stacking subderivation of length $k_1$ in $U^{\alpha'}_{j} \longrightarrow U^{\alpha'}_{j+k_1 \cdot D_1}$. By Lemma 3.32 it follows that both $U^{\alpha}_{j+D_1 \cdot k_1}$ are unmarked.

(1) Let $j_1 = j + D_1 \cdot k_1$ and let us show that there exists some $i \geq j_1$ such that $T_B$ occurs at $i + k_1 + 1$.

If such an $i$ does not exist, then for every $\alpha \in \{-, +\}$, the infinite derivation

$$U^{\alpha}_{j_1} \longrightarrow U^{\alpha}_{j_1+1} \longrightarrow \cdots$$

does not contain any stacking sequence of length $k_1$. By Lemma 3.31 we would have

$$\forall k \geq j_1, \quad \|U^{\alpha}_{k}\| \leq \|U^{\alpha}_{j_1}\| + k_1 \cdot K_0.$$

As the set $\{\|U^{\alpha}_{k}\|, k \geq j_1, \alpha \in \{-, +\}\}$ is finite, there would be a repetition

$$(U^-_k, U^+_k) = (U^-_{k'}, U^+_{k'}) \quad \text{with } j_1 \leq k < k' \text{ and } \pi_k < \pi_{k'},$$

so that $T_{cut}$ would have been defined on some finite prefix of the branch, contradicting the hypothesis that the branch is infinite.

(2) Let $i > i'$ be the smallest integer (in $[j_1, \infty[$ ) fulfilling point (1) above, and suppose that $T_B^\alpha$ occurs at $i + k_1 + 1$.

By Lemma 8.4,

$$\forall \ell \in [j_1, i], \quad \|U_\ell^{-\alpha}\| \geq N_0 - k_2 > D_2.$$

Using Lemma 8.3, Lemma 3.29, and inequality (8.1) we conclude that

$$\forall \ell \in [j_1, i], \quad U_\ell^{-\alpha} \text{ is loop-free.}$$

By an argument analogous to that used in Lemma 8.3 we see that $U_{j_1}^{-\alpha} = S_0 \odot u$ for some $|u| \leq (j_1 - i_0)$, and by Lemma 3.13 we get

$$\|U_{j_1}^{-\alpha}\| \leq (j_1 - i_0) \cdot K_0 + \|S_0\|. \tag{8.19}$$

We also know that

$$\|S_0\| \leq \|U_i^{-\alpha}\| \leq \|U_{i-1}^{-\alpha}\| + K_0. \tag{8.20}$$

As the derivation $U_{j_1}^{-\alpha} \longrightarrow U_{i-1}^{-\alpha}$ contains no stacking subderivation of length $k_1$ and consists of loop-free series only, by Lemma 3.30 we obtain

$$\|U_{i-1}^{-\alpha}\| \leq \|U_{j_1}^{-\alpha}\| - (i - j_1 - 2)/k_1. \tag{8.21}$$

Combining the three inequalities (8.19), (8.20), (8.21) we get successively

$$\|S_0\| \leq \|S_0\| + (j_1 - i_0 + 1) \cdot K_0 - (i - j_1 - 2)/k_1,$$

$$(i - j_1 - 2) \leq (j_1 - i_0 + 1) \cdot k_1 K_0,$$

(8.22)
$$\begin{aligned}(i - i') &= (i - j_1 - 2) + (j_1 - i' + 2) \leq (j_1 - i_0 + 1) \cdot k_1 \cdot K_0 + D_1 \cdot k_1 + k_1 + 3\\ &= (i' - i_0) \cdot k_1 \cdot K_0 + k_1^2 \cdot D_1 \cdot K_0 + k_1^2 \cdot K_0 + 2 \cdot k_1 \cdot K_0 + D_1 \cdot k_1 + k_1 + 3\\ &= (K_1 - 1)(i' - i_0) + K_2 - k_1 - 1.\end{aligned}$$

(3) By the choice of $i', i$, we know that $i' \leq i_0 + L \leq i$. Using (8.22) we obtain

$$i \leq i' + (K_1 - 1)(i' - i_0) + K_2 - k_1 - 1,$$

$$i \leq i_0 + K_1 \cdot L + K_2 - k_1 - 1.$$

Assertion (8.18) is now established for Case 2 as well as for Case 1.

From (8.18) and Lemma 8.6 the lemma follows.

(We illustrate our argument in Figure 5.) $\square$

Let us give now a stronger version of Lemma 8.7 in which we analyze the *size of the coefficients* of the linear combinations whose existence is proved in Lemma 8.7. We recall that

$$K_3 = K_0|Q|, \quad K_4 = D_1.$$

Let us fix a total ordering on $\mathcal{G}_0$:

$$\mathcal{G}_0 = \{\theta_1, \theta_2, \ldots, \theta_d\}, \text{ where } d = \mathrm{Card}(\mathcal{G}_0).$$

Let us remark that $d \leq \mathrm{Card}(X^{\leq k_4}) = d_0$.

LEMMA 8.8. *Let $L \geq 0$. There exists $i \in [i_0 + L, i_0 + K_1 \cdot L + K_2]$ and, for every $\alpha \in \{-, +\}$, there exists a deterministic rational family $(\beta_{i,j}^\alpha)_{1 \leq j \leq d}$ fulfilling*

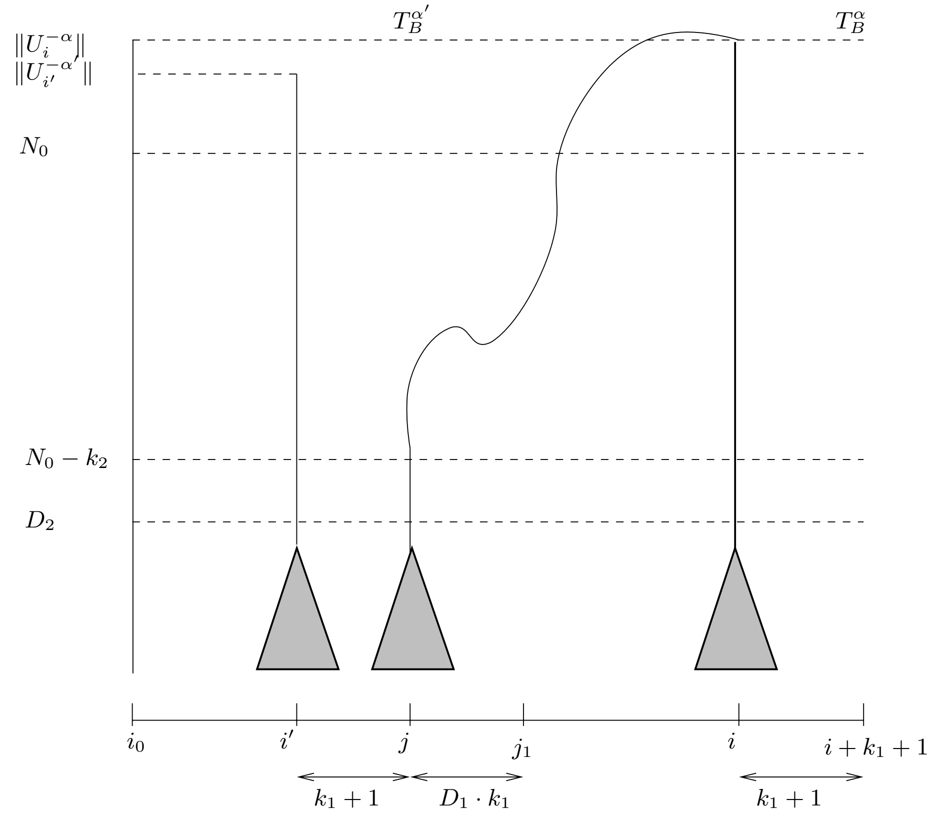


FIG. 5. *Two successive $T_B$.*

(1) $U_i^{\alpha} = \sum_{j=1}^{d} \beta_{i,j}^{\alpha} \cdot \theta_j$,
(2) $\|\beta_{i,*}^{\alpha}\| \le K_3 \cdot (i - i_0) + K_4$.

*Proof.* By Lemma 8.7 there exists $i \in [i_0 + L, i_0 + K_1 \cdot L + K_2]$ and $\alpha \in \{-, +\}$ such that $T_B^{\alpha}$ occurs at $i$. Let us use the notation of the proof of Lemma 8.6 and compute upper-bounds on the coefficients of $U_i^{-\alpha}, U_i^{\alpha}$ expressed as linear combinations of the vectors of $\mathcal{G}_0$.

*Coefficients of $U_i^{-\alpha}$.* $U_i^{-\alpha} = U_{i-k_1-1}^{-\alpha} \odot u'$ for some $u' \in X^*$, $|u'| = k_1$. By Lemma 3.15, $U_i^{-\alpha}$ can be expressed in one of the two following forms:

$$U_i^{-\alpha} = S_0 \odot (\bar{u}_k u''), \text{ where } u'' \text{ is a suffix of } u', \tag{8.23}$$

$$U_i^{-\alpha} = \sum_{k=1}^{q} (E_k \odot u_2 u') \cdot (S_0 \odot \bar{u}_k). \tag{8.24}$$

In case (8.23) we can choose as vector of coordinates: $\beta_{i,\star}^{-\alpha} = \epsilon_{j_0}^{d}$. We then have $\|\beta_{i,\star}\| = 2 \le K_4$.

In case (8.24), we can choose $\beta_{i,\star}^{-\alpha} = E \odot u_2 u'$ (completed with $\emptyset$ in all the columns $j$ not corresponding to some vector $S_0 \odot \bar{u}_k$ of $\mathcal{G}_0$). We then have

$$\|\beta_{i,\star}\| = \|E \odot u_2 u'\| \le K_0 \cdot (i - i_0) \le K_3 \cdot (i - i_0).$$

*Coefficients of $U_i^\alpha$.* By the definition of $T_B^\alpha$

$$U_i^\alpha = \sum_{\ell=1}^{r} \tau_\ell \cdot (U_{i-k_1-1}^{-\alpha} \odot \bar{w}_\ell), \tag{8.25}$$

where $\|\tau\| \leq D_1$, $|\bar{w}_\ell| \leq k_0$.

Replacing $u'$ by $\bar{w}_\ell$ in the above analysis, we get

$$\forall \ell \in [1, r], \quad U_{i-k_1-1}^{-\alpha} \odot \bar{w}_\ell = \sum_{j=1}^{d} \gamma_{\ell,j} \cdot \theta_j, \tag{8.26}$$

with $\|\gamma_{\ell,\star}\| \leq K_0 \cdot (i - i_0)$.

Equalities (8.25), (8.26) show that

$$U_i^\alpha = \tau \cdot \gamma \cdot \theta,$$

where $\tau$, $\gamma$, $\theta$ are deterministic rational matrices of dimensions $(1, r)$, $(r, d)$, $(d, 1)$, respectively. Let us choose $\beta_{i,\star} = (\tau \cdot \gamma)$.

$$\|\beta_{i,\star}\| \leq \|\tau\| + \|\gamma\| \leq D_1 + r \cdot K_0 \cdot (i - i_0)$$
$$\leq D_1 + |Q| \cdot K_0 \cdot (i - i_0) = K_3 \cdot (i - i_0) + K_4. \qquad \square$$

Lemma 8.9. *There exists $i_0 \leq \kappa_1 < \kappa_2 < \cdots < \kappa_d$ and deterministic rational vectors $(\beta_{i,j}^\alpha)_{1 \leq j \leq d}$ (for every $i \in [1, d]$) such that*

(0) $W(\kappa_1) \geq 1$,

(1) $\forall i, \forall \alpha, U_{\kappa_i}^\alpha = \sum_{j=1}^{d} \beta_{i,j}^\alpha \theta_j \in V_0$,

(2) $\forall i, \forall \alpha, \|\beta_{i,*}^\alpha\| \leq s_i$,

(3) $\forall i, W(\kappa_{i+1}) - W(\kappa_i) \geq \delta_{i+1}$,

*where the sequences $(\delta_i, \ell_i, L_i, s_i, S_i, \sigma_i)$ are those defined by relations* (6.8), (6.9) *in section* 6.

*Proof.* Let us consider the additional property

(4) $\kappa_i - i_0 \leq L_i$.

We prove by induction on $i$ the conjunction (1) $\wedge$ (2) $\wedge$ (3) $\wedge$ (4).

$\boldsymbol{i = 1}$**:** By Lemma 8.8, there exists $\kappa_1 \in [i_0, i_0 + K_2]$ such that $\forall \alpha \in \{-, +\}$, there exists a deterministic vector $(\beta_{1,j}^\alpha)_{1 \leq j \leq d}$, such that

$$U_{\kappa_1}^\alpha = \sum_{j=1}^{d} \beta_{1,j}^\alpha \theta_j,$$

and in addition $\|\beta_{1,*}^\alpha\| \leq K_3 K_2 + K_4 = s_1$.

$\boldsymbol{i \longrightarrow i + 1}$**:** Suppose that $\kappa_1 < \kappa_2 < \cdots < \kappa_i$ fulfill (1) $\wedge$ (2) $\wedge$ (3) $\wedge$ (4). By Lemma 8.8, there exists $\kappa_{i+1} \in [i_0 + L_i + \ell_{i+1}, i_0 + K_1(L_i + \ell_{i+1}) + K_2]$ such that $\forall \alpha \in \{-, +\}$, there exists a deterministic vector $(\beta_{i+1,j}^\alpha)_{1 \leq j \leq d}$, such that

$$U_{\kappa_{i+1}}^\alpha = \sum_{j=1}^{d} \beta_{i+1,j}^\alpha \theta_j, \tag{8.27}$$

and in addition

$$\begin{aligned} \|\beta_{i+1,*}^\alpha\| \leq K_3(K_1(L_i + \ell_{i+1}) + K_2) + K_4 &= K_3 L_{i+1} + K_4 \\ &= s_{i+1}. \end{aligned} \tag{8.28}$$

By Lemma 8.2

$$2(W(\kappa_{i+1}) - W(\kappa_i)) + 3 \geq \kappa_{i+1} - \kappa_i \geq \ell_{i+1} = 2\delta_{i+1} + 3,$$

and hence

$$W(\kappa_{i+1}) - W(\kappa_i) \geq \delta_{i+1}. \tag{8.29}$$

Finally,

$$\kappa_{i+1} - i_0 \leq K_1(L_i + l_{i+1}) + K_2 = L_{i+1}. \tag{8.30}$$

The above properties (8.27)–(8.30) prove the required conjunction.

It remains to prove point (0): the integer $\kappa_1$ introduced by Lemma 8.8 is such that $T_B$ occurs at $\kappa_1$, and hence

$$\begin{aligned} W(\kappa_1) &= W(\kappa_1 - k_1 - 1) + k_1 - 1 \\ &\geq W(\kappa_1 - k_1 - 1) + 2 \geq 1. \qquad \square \end{aligned}$$

LEMMA 8.10. *Let $(x_i)_{i\in\mathbb{N}}$ be an infinite branch of $\tau$. Then there exist some $i_0 \in \mathbb{N}$ such that $(x_i)_{i\geq i_0}$ is a B-stacking sequence.*

*Proof.* Let us distinguish, a priori, several cases, and see that only the case where $\tau$ admits a B-stacking sequence is possible.

*Case* 1. *$T_B$ occurs finitely often on $\tau$.* Let $j$ be the largest integer such that $T_B$ occurs at $j$. By the arguments used in the proof of Lemma 8.7, Case 2, we know that $U^-_{j+k_1\cdot D_1}, U^+_{j+k_1\cdot D_1}$ are both unmarked, and that

$$\forall k \geq j + k_1 \cdot D_1, \forall \alpha \in \{-,+\}, \quad \|U^\alpha_k\| \leq \|U^\alpha_{j+k_1\cdot D_1}\| + k_1 \cdot K_0.$$

This would imply that the branch contains a finite prefix on which $T_{cut}$ is defined, which is impossible on an infinite branch.

*Case* 2. *For some sign $\alpha$, there are infinitely many integers $i$ such that $[T^\alpha_B$ occurs at $i + k_1 + 1$ and $\|U^{-\alpha}_i\| < N_0]$.* In this case there would exist an infinite sequence of integers $i_1 < i_2 < \cdots < i_\ell <$ such that

$$\forall \ell \geq 0, \quad U^{-\alpha}_{i_1} = U^{-\alpha}_{i_\ell}.$$

For a given $U^{-\alpha}_i$, only a finite number of values are possible for the pair $(U^-_{i+k_1+1}, U^+_{i+k_1+1})$. Hence there exist integers $\ell < \ell'$ such that

$$\ell < \ell', \quad \pi_\ell < \pi_{\ell'}, \quad \text{and} \quad (U^-_{\ell+k_1+1}, U^+_{\ell+k_1+1}) = (U^-_{\ell'+k_1+1}, U^+_{\ell'+k_1+1}).$$

Here again $T_{cut}$ would have a nonempty value on some prefix of $\tau$, which is impossible.

*Case* 3. *$T_B$ occurs infinitely often on $\tau$ and, for every sign $\alpha$, there are only finitely many integers $i$ such that $[T^\alpha_B$ occurs at $i + k_1 + 1$ and $\|U^{-\alpha}_i\| < N_0]$.*

Let us consider the set $I_0$ of the integers $i$ such that there exists a sign $\alpha_i$ such that

$$[T^{\alpha_i}_B \text{ occurs at } i + k_1 + 1 \text{ and } \|U^{-\alpha_i}_i\| \geq N_0].$$

By the hypothesis of Case 3, $I_0 \neq \emptyset$. Let $i_0$ be such that

$$\|U^{-\alpha_{i_0}}_{i_0}\| = \min\{\|U^{-\alpha_i}_i\| \mid i \in I_0\}.$$

Then $(x_i)_{i\geq i_0}$ is a B-stacking sequence. $\square$

## 9. Termination.

LEMMA 9.1. *$\hat{\mathcal{S}}_{ABC}$ is terminating on every unmarked assertion $A_0$: if $A_0 \in \mathcal{A}$ is unmarked, then $\exists n_0 \in \mathbb{N}$, $\hat{\mathcal{S}}_{ABC}^{n_0+1}(A_0) = \hat{\mathcal{S}}_{ABC}^{n_0}(A_0)$.*

*Proof.* Suppose $A_0 \in \mathcal{A}$, $A_0$ is true, $A_0$ is unmarked, and

$$\forall n \in \mathbb{N}, \quad \hat{\mathcal{S}}_{ABC}^{n}(A_0) \prec \hat{\mathcal{S}}_{ABC}^{n+1}(A_0). \tag{9.1}$$

Let us consider all the constants associated to this precise $A_0$, the equivalence $\bar{\psi}$, and the dpda $\mathcal{M}$ in section 6. Let us note that $t_n = \hat{\mathcal{S}}_{ABC}^{n}(A_0)$ (for every $n \in \mathbb{N}$) and let

$$t_\infty = \text{least upper-bound}\{t_n \mid n \in \mathbb{N}\}.$$

Let us note that, by definition (7.10), the strict inequality (9.1) implies that

$$\forall n \in \mathbb{N}, \quad t_n \text{ is consistent.} \tag{9.2}$$

Let us denote by $x_n$ the node of $t_n$ such that $t_{n+1} = t_n[\hat{\Delta}(t_n)/x_n]$. Let us notice that as every $x_n$ is unclosed in $t_n$, one can prove by induction that every $t_n$ is repetition-free. Hence

$$t_\infty \text{ is repetition-free.} \tag{9.3}$$

By Koenig's lemma, $t_\infty$ contains an infinite branch $y_0 y_1 \cdots y_s \cdots$ whose (infinite) labeling word is $A_0 A_1 \cdots A_s \cdots$ (where $A_s = t_\infty(y_s)$).

Condition (C3) in the definition of $T_C^{(\mathcal{O})}$, combined with Lemma 3.17, shows that every equation $(\pi, T, U)$ produced by $T_C$ has size

$$\max\{\|T\|, \|U\|\} \leq D_2, \tag{9.4}$$

and hence that the number of possible unweighted equations produced by $T_C$ is *finite*. Hence $T_C$ occurs only a finite number of times on this branch (because $t_\infty$ is repetition-free (9.3) and $T_{cut}$ cannot occur on an infinite branch). Let $n_0$ be the last point where $T_C$ occurs (or $n_0 = 0$ if $T_C$ never occurs on this branch). $(y_{n_0+i})_{i \geq 0}$ is a branch of a tree $t' \in \mathcal{T}(\mathcal{S}_{AB}, A_{n_0})$. Let us notice also that

$$\text{every equation produced by } T_C \text{ is unmarked} \tag{9.5}$$

(by condition (C4) in the definition of $T_C^{(\mathcal{O})}$; see section 7), and

$$\text{every equation produced by } T_C \text{ has a length } \lambda \leq \lambda_2, \tag{9.6}$$

because it has a length $\leq d_0$ and $d_0 \leq \lambda_2$ by definition (6.11) in section 6. Moreover, the root $A_0$ of $t_\infty$ must have a size $\leq D_2$ (by definition (6.10) in section 6), must be unmarked (by the hypothesis of the lemma), and must have a length $\lambda_0 \leq \lambda_2$ (by definition (6.11) in section 6). Hence, in either case, $t'$ fulfills the hypotheses (8.1) and (8.2) stated in section 3.3 and assumed in section 8.

As $\mathcal{S}_{ABC}$ is a strategy for $\mathcal{B}_0$ and $A_0$ is true, $A_{n_0}$ is also true, and hence hypothesis (8.3) assumed in section 8 is fulfilled. We may now apply the results obtained in section 8.2.

By Lemma 8.10, the branch $(y_{n_0+i})_{i \geq 0}$ must contain an infinite B-stacking sequence. Let us remark that, as $T_\emptyset$ does not occur (otherwise the branch would be finite), every equation $(\pi, U^-, U^+)$ labeling this branch is such that $U^- \neq \emptyset$, $U^+ \neq \emptyset$.

By Lemma 8.9 such a B-stacking sequence contains a subsequence $(A_{\kappa_1}, A_{\kappa_2}, \dots, A_{\kappa_d})$ with $d \leq d_0$, fulfilling hypotheses (1), (2) of Lemma 5.5, and by the above remark it fulfills hypothesis (5.6) of section 5.2 too. Let $n_i \in \mathbb{N}$ such that $x_{n_i} = y_{\kappa_i}$ for $1 \leq i \leq d$. By (9.2), $\Omega(\bar{\Pi}(t_{n_d}), \mathrm{im}(t_{n_d})) \neq \emptyset$. Let us consider some

$$\mathcal{O} \in \Omega(\bar{\Pi}(t_{n_d}), \mathrm{im}(t_{n_d})).$$

Let $\mathcal{S}_d = (A_{\kappa_i})_{1 \leq i \leq d}$ and $D = \mathrm{D}^{(\mathcal{O})}(\mathcal{S}_d)$. By Lemma 5.5,

$$(9.7) \qquad \mathrm{INV}^{(\mathcal{O})}(\mathcal{S}_d) \neq \perp, \quad D \in [0, d-1], \quad \text{and} \quad \|| \mathrm{INV}^{(\mathcal{O})}(\mathcal{S}_d) \|| \leq \Sigma_{d_0} + s_{d_0}.$$

Let $\mathcal{S}_{D+1} = (A_{\kappa_i})_{1 \leq i \leq D+1}$. By hypothesis (2) of Lemma 5.5 (we established that this hypothesis is true),

$$\bar{\Pi}(t_{n_{D+1}}) \leq \bar{\Pi}(t_{n_d}),$$

and it is straightforward that

$$\mathrm{im}(t_{n_{D+1}}) \subseteq \mathrm{im}(t_{n_d});$$

hence,

$$(9.8) \qquad \mathcal{O} \in \Omega(\bar{\Pi}(t_{n_{D+1}}), \mathrm{im}(t_{n_{D+1}})).$$

Let $W_{D+1} = A_0 \cdot A_1 \cdots A_{\kappa_1} \cdots A_{\kappa_{D+1}}$ (the word from the root to $y_{\kappa_{D+1}}$). Let us notice that

$$(9.9) \qquad \mathrm{D}^{(\mathcal{O})}(\mathcal{S}_{D+1}) = \mathrm{D}^{(\mathcal{O})}(\mathcal{S}_d) = D, \quad \mathrm{INV}^{(\mathcal{O})}(\mathcal{S}_{D+1}) = \mathrm{INV}^{(\mathcal{O})}(\mathcal{S}_d).$$

By (9.7), (9.9),

$$(9.10) \qquad \rho_e(\mathrm{INV}^{(\mathcal{O})}(\mathcal{S}_{D+1})) \in T_C^{(\mathcal{O})}(W_{D+1}).$$

By (9.8), (9.10), the set $\{\mathcal{O} \in \Omega(\bar{\Pi}(t_{n_{D+1}}), \mathrm{im}(t_{n_{D+1}})), T_C^{(\mathcal{O})}(W_{D+1}) \neq \emptyset\}$ is not empty, so that case (0) of the definition of $\hat{\Delta}(t)$ (see section 7) is fulfilled and

$$\hat{\Delta}(t_{n_{D+1}}) = A_{n_{D+1}}(T_C^{(\mathcal{O}_0)}(W_{D+1})),$$

i.e., $T_C$ occurs at $y_{\kappa_{D+1}+1}$. This is a contradiction with the minimality of $n_0$. We have proved that hypothesis (9.1) is impossible. Hence the lemma is proved. $\square$

## 10. Elimination.

**10.1. System $\mathcal{B}_1$.** We prove here that the new formal system $\mathcal{B}_1$ obtained by *elimination* of metarule (R5) in $\mathcal{B}_0$ is recursively enumerable and complete. The decidability of the bisimulation problem follows.

Let $\mathcal{B}_1 = \langle \mathcal{A}, H, \mid\!\!\!-\!\!-_{\mathcal{B}_1} \rangle$, where $\mathcal{A}, H$ are the same as in $\mathcal{B}_0$, but the *elementary deduction relation* $|\!\mid\!\!\!-\!\!-_{\mathcal{B}_1}$ is the relation generated by the subset of metarules (R0), (R1), (R2), (R3), (R′3), (R4), (R6), (R7), (R8), i.e., all the metarules of $\mathcal{B}_0$ except (R5). The deduction relation $\mid\!\!\!-\!\!-_{\mathcal{B}_1}$ is now defined by

$$\mid\!\!\!-\!\!-_{\mathcal{B}_1} = \overset{\langle * \rangle}{|\!\mid\!\!\!-\!\!-}_{\mathcal{B}_1} \circ \overset{[1]}{|\!\mid\!\!\!-\!\!-}_{R0,R3,R'3,R4} \circ \overset{\langle * \rangle}{|\!\mid\!\!\!-\!\!-}_{\mathcal{B}_1}.$$

LEMMA 10.1. *$\mathcal{B}_1$ is a deduction system.*

*Sketch of proof.* As $\mid\!\!\!-\!\!-_{\mathcal{B}_1} \subseteq \mid\!\!\!-\!\!-_{\mathcal{B}_0}$, property (A1) is fulfilled by $\mid\!\!\!-\!\!-_{\mathcal{B}_1}$.

By the well-known decidability properties for finite-automata, rules (R0), (R1), (R2), (R3), (R′3), (R4), (R6), (R7), (R8) are recursively enumerable. Hence property (A2) is fulfilled by $\mathcal{B}_1$. $\square$

**Completeness.**

Definition 10.2. *Let $P$ be a finite subset of $\mathcal{A}$ and let $\bar{\pi} \in \mathbb{N}$. $P$ is said to be* locally $\bar{\pi}$-consistent *iff, for every $(\pi, S, S') \in P$, if*

$$\pi < \bar{\pi},$$

*then there exists $\mathcal{R}_1 \in \bar{\mathcal{B}}_1$ such that*

$$[\pi, S, S', \mathcal{R}_1] \subseteq \mathrm{Cong}(P).$$

Lemma 10.3. *Let $P$ be a finite subset of $\mathcal{A}$ and let $\bar{\pi} \in \mathbb{N}$. If $P$ is locally $\bar{\pi}$-consistent, then $P$ is $\bar{\pi}$-consistent.*

*Proof.* Let us consider, for every integer $n \geq 0$ $p \geq 0$, the following property $\mathcal{Q}(n,p)$: $\forall \pi \in \mathbb{N}$, $\lambda \in \mathbb{N} - \{0\}$, $S, S' \in \mathsf{DRB}_{1,\lambda}\langle\langle\, V\,\rangle\rangle$,

$$(\pi, S, S') \in \mathrm{Cong}_p(P) \quad \text{and} \quad \pi + n - 1 < \bar{\pi} \Rightarrow$$
$$\exists \mathcal{R}_n \in \mathcal{B}_n(S, S'), \quad [\pi, S, S', \mathcal{R}_n] \subseteq \mathrm{Cong}(P). \tag{10.1}$$

Let us prove by lexicographic induction on $(n, p)$ that

$$\forall (n,p) \in \mathbb{N} \times \mathbb{N}, \quad \mathcal{Q}(n,p). \tag{10.2}$$

***n* = 0, *p* = 0:** The only possible value of $\mathcal{R}_0 \in \mathcal{B}_0(S, S')$ is $\mathcal{R}_0 = \{(\epsilon, \epsilon)\}$, and $[\pi, S, S', \mathcal{R}_0] = \{(\pi, S, S')\} \subseteq \mathrm{Cong}_0(P)$.

***p* > 0:** There exists a subset $Q \subseteq \mathcal{P}_f(\mathcal{A})$, such that

$$P \overset{\langle p-1\rangle}{||\!\!-\!\!-}_{\mathcal{C}}\, Q \quad \text{and} \quad Q \overset{\langle 1\rangle}{||\!\!-\!\!-}_{\mathcal{C}}\, \{(\pi, S, S')\}.$$

As every rule of $\mathcal{B}_0$ increases the weight, we can suppose that every assertion of $Q$ has a weight $\leq \pi$. Hence, by the induction hypothesis,

$$\forall (\pi', T, T') \in Q,\ \exists \mathcal{R}_n \in \mathcal{B}_n(T, T'), \quad [\pi', T, T', \mathcal{R}_n] \subseteq \mathrm{Cong}(P). \tag{10.3}$$

Let us consider the type of rule used in the last step, $Q \overset{\langle 1\rangle}{||\!\!-\!\!-}_{\mathcal{C}} \{(\pi, S, S')\}$, of the above deduction.

(R0): $(\pi - 1, S, S') \in Q$. By (10.3), $\exists \mathcal{R}_n \in \mathcal{B}_n(S, S')$,

$$[\pi - 1, S, S', \mathcal{R}_n] \subseteq \mathrm{Cong}(P).$$

As $[\pi - 1, S, S', \mathcal{R}_n] \overset{\langle 1\rangle}{||\!\!-\!\!-}_{\mathcal{C}} [\pi, S, S', \mathcal{R}_n]$,

$$[\pi, S, S', \mathcal{R}_n] \subseteq \mathrm{Cong}(P).$$

(R1): $(\pi, S', S) \in Q$ (analogous to the above case).

(R2): $(\pi, S, T), (\pi, T, S') \in Q$. By (10.3), $\exists \mathcal{R}_n \in \mathcal{B}_n(S, T)$, $\mathcal{R}'_n \in \mathcal{B}_n(T, S')$,

$$[\pi, S, T, \mathcal{R}_n] \subseteq \mathrm{Cong}(P), [\pi, T, S', \mathcal{R}'_n] \subseteq \mathrm{Cong}(P).$$

Using the properties mentioned in section 4.4, we get that

$$[\pi, S, S', \mathcal{R}_n \circ \mathcal{R}'_n] \subseteq \mathrm{Cong}(P).$$

(R3): In this case, $\mathcal{R}_n = \mathrm{Id} \cap X^{\leq n} \times X^{\leq n} \in \mathcal{B}_n(S, S')$, and

$$[\pi, S, S', \mathcal{R}_n] \subseteq \{(\pi, S, S')\} \cup \{(\pi + k, T, T), 1 \leq k \leq n, T \in \mathsf{DRB}_{1,\lambda}\langle\langle\ V\ \rangle\rangle\} \\ \subseteq \mathrm{Cong}(P).$$

(R′3): In this case, $\mathcal{R}_n = \mathrm{Id} \cap X^{\leq n} \times X^{\leq n} \in \mathcal{B}_n(S, S')$, and

$$[\pi, S, S', \mathcal{R}_n] = \{(\pi + k, S \odot u, \rho_e(S) \odot u) \mid 0 \leq k \leq n, u \in X^k\} \subseteq \mathrm{Cong}(P)$$

(because $\rho_e(S) \odot u = \rho_e(S \odot u)$).

(R6): $(\pi, S_1 \cdot S' + U, S') \in Q$, $S = S_1^* \cdot U$. By (10.3), $\exists \mathcal{R}_n \in \mathcal{B}_n(S_1 \cdot S' + U, S')$,

$$[\pi, S_1 \cdot S' + U, S', \mathcal{R}_n] \subseteq \mathrm{Cong}(P).$$

Using the properties mentioned in section 4.4, we get that

$$\begin{aligned}[\pi, S, S', \mathcal{R}_n^{\langle S_1, *\rangle}] &= [\pi, S_1^* \cdot U, S', \mathcal{R}_n^{\langle S_1, *\rangle}] \\ &\subseteq \mathrm{Cong}[\pi, S_1 \cdot S' + U, S', \mathcal{R}_n] \\ &\subseteq \mathrm{Cong}(Q) \subseteq \mathrm{Cong}(P).\end{aligned}$$

(R7): $(\pi, S_1, S_1') \in Q$, $S = S_1 \cdot T$, $S' = S_1' \cdot T$. By (10.3), $\exists \mathcal{R}_n \in \mathcal{B}_n(S_1, S_1')$, $[\pi, S_1, S_1', \mathcal{R}_n] \subseteq \mathrm{Cong}(P)$. Using the properties mentioned in section 4.4, we get that

$$\begin{aligned}[\pi, S, S', \langle S_1 | \mathcal{R}_n\rangle] &= [\pi, S_1 \cdot T, S_1' \cdot T, \langle S_1 | \mathcal{R}_n\rangle] \\ &\subseteq \mathrm{Cong}([\pi, S_1, S_1', \mathcal{R}_n]) \\ &\subseteq \mathrm{Cong}(Q) \subseteq \mathrm{Cong}(P).\end{aligned}$$

(R8): $\forall i \in [1, \delta]$, $(\pi, T_{i,*}, T'_{i,*}) \in Q$, $S = S_1 \cdot T$, $S' = S_1 \cdot T'$. By (10.3), $\exists \mathcal{R}_{1,n}, \ldots, \mathcal{R}_{\delta,n} \in \mathcal{B}_n(T_{i,*}, T'_{i,*})$, such that

$$[\pi, T_{i,*}, T'_{i,*}, \mathcal{R}_{i,n}] \subseteq \mathrm{Cong}(P).$$

Using the properties mentioned in section 4.4, we get that

$$\begin{aligned}[\pi, S, S', \langle S, \mathcal{R}_{*,n}\rangle] &= [\pi, S_1 \cdot T, S_1 \cdot T', \langle S, \mathcal{R}_{*,n}\rangle] \\ &\subseteq \mathrm{Cong}\left(\bigcup_{1 \leq i \leq \delta} [\pi, T_{i,*}, T'_{i,*}, \mathcal{R}_{i,n}]\right) \\ &\subseteq \mathrm{Cong}(Q) \subseteq \mathrm{Cong}(P).\end{aligned}$$

In all cases $\mathcal{Q}(n, p)$ has been established.

***n* > 0, *p* = 0:** $(\pi, S, S') \in P$. As $P$ is locally $\bar{\pi}$-consistent and $\pi \leq \pi + n - 1 < \bar{\pi}$, there exist $\mathcal{R}_1 \in \mathcal{B}_1(S, S')$, $q \in \mathbb{N}$ such that

$$[\pi, S, S', \mathcal{R}_1] \subseteq \mathrm{Cong}_q(P). \tag{10.4}$$

As $(n-1, q) < (n, 0)$, by the induction hypothesis, $\forall (x, x') \in \mathcal{R}_1 \cap X \times X$, $\exists \mathcal{R}_{x,x',n-1} \in \mathcal{B}_{n-1}(S \odot x, S' \odot x')$ such that

$$[\pi + 1, S \odot x, S' \odot x', \mathcal{R}_{x,x',n-1}] \subseteq \mathrm{Cong}(P). \tag{10.5}$$

Let us consider $\mathcal{R}_n = \{(\epsilon,\epsilon)\} \bigcup_{(x,x') \in \mathcal{R}_1 \cap X \times X} \{(x,x')\} \cdot \mathcal{R}_{x,x',n-1}$. One can check that $\mathcal{R}_n \in \mathcal{B}_n(S,S')$ and, by (10.4), (10.5), we obtain

$$[\pi, S, S', \mathcal{R}_n] = \{(\pi,S,S')\} \bigcup_{(x,x') \in \mathcal{R}_1 \cap X \times X} [S \odot x, S' \odot x', \mathcal{R}_{x,x',n-1}] \subseteq \mathrm{Cong}(P).$$

Let us define now an oracle $\mathcal{O} \in \Omega$ which is $\bar{\pi}$-consistent with $P$. For every $(S,S') \in \bigcup_{\lambda \geq 1} \mathsf{DRB}_{1,\lambda}\langle\langle\ V\ \rangle\rangle \times \mathsf{DRB}_{1,\lambda}\langle\langle\ V\ \rangle\rangle$ occurring in $\mathrm{Cong}(P)$ (i.e., as the last two components of an assertion in $\mathrm{Cong}(P)$), let us note the following:

$$\begin{aligned} W(S,S') &= \min\{\pi \in \mathbb{N} \mid (\pi,S,S') \in \mathrm{Cong}(P)\},\\ D(S,S') &= \max\{\bar{\pi} - W(S,S'), 0\},\\ C(S,S') &= \min\{\mathcal{R} \in \mathcal{B}_{D(S,S')}(S,S') \mid [W(S,S'),S,S',\mathcal{R}] \subseteq \mathrm{Cong}(P)\}. \end{aligned}$$

Notice that $C(S,S')$ is well-defined, owing to property (10.2). We then define $\mathcal{O}$ as follows: for every $(S,S')$ occurring in $\mathrm{Cong}(P)$,

$$\mathcal{O}(S,S') = \min\{\mathcal{R} \in \mathcal{B}_\infty(S,S') \mid C(S,S') = \mathcal{R} \cap (X^{\leq D(S,S')} \times X^{\leq D(S,S')})\}, \tag{10.6}$$

and for every $(S,S')$ not occurring in $\mathrm{Cong}(P)$,

$$\begin{aligned} &\mathcal{O}(S,S') = \min\{\mathcal{R} \in \mathcal{B}_\infty(S,S')\} \ \text{(if } S \sim S'),\\ &\mathcal{O}(S,S') = \mathrm{Id}_{X^*} \ \text{(if } S \not\sim S'). \end{aligned} \tag{10.7}$$

One can check that, by the choice of $C(S,S')$, $\mathcal{O}$ is $\bar{\pi}$-consistent with $P$. □

LEMMA 10.4. *Let $A_0 \in \mathcal{A}$ such that $H(A_0) = \infty$. Let us consider the sequence of trees $t_n = \hat{\mathcal{S}}^n_{ABC}(A_0)$. For every integer $n \geq 0$, $t_n$ is consistent.*

Let us say that the strategy $T$ "applies to" node $x$ iff $x$ has exactly $m$ sons $x \cdot 1, x \cdot 2, \dots, x \cdot m$ and

$$\tau(x1) \cdot \tau(x \cdot 2) \cdots \tau(x \cdot m) \in T(\tau(x[0]) \cdot \tau(x[1]) \cdots \tau(x[|x|]));$$

i.e., the word consisting of the labels of the sons of $x$ belongs to the image of the path from $\epsilon$ (included) to $x$ (included) by the strategy $T$.

*Proof.* For every $k \in \mathbb{N}$ we define

$$\bar{\pi}_k = \bar{\Pi}(t_k).$$

We prove by induction on $(n,\pi)$ the following property $\mathcal{R}(n,\pi)$:

$$\forall x \in \mathrm{dom}(t_n), \text{ if } t_n(x) = (\pi,S,S') \text{ with } \pi < \bar{\pi}_n, \text{ then} \tag{10.8}$$

$$\exists \mathcal{R}_1 \in \mathcal{B}_1(S,S'), [\pi,S,S',\mathcal{R}_1] \subseteq \mathrm{Cong}(\mathrm{im}(t_n)). \tag{10.9}$$

At every step of our proof by induction, we consider some node $x$ of $t_n$ fulfilling hypothesis (10.8) and show that it must fulfill (10.9). Let us notice that if $x$ is not closed, then hypothesis (10.8) cannot be true, by minimality of $\bar{\pi}_n$. Let us notice also that if $x$ is closed, but there is some $x' \prec x$ such that $t_n(x') = t_n(x)$, then (10.9) on $x$ is the same property as (10.9) for $x'$. Hence, in what follows, we can suppose that $x$ is closed and that it is minimal (w.r.t. $\preceq$):

$$x = \min_{\preceq}\{y \in \mathrm{dom}(t_n) \mid t_n(y) = t_n(x)\}. \tag{10.10}$$

***$n = 0$, $\pi = 0$:*** $\mathrm{dom}(t_0) = \{\epsilon\}$, $t_0(\epsilon) = A_0$. If $\epsilon$ is not closed, then $\bar{\pi}_0 = \pi = 0$, and hence there is no node $x$ fulfilling hypothesis (10.8). Otherwise, $\bar{\pi}_0 = \infty$ and $x = \epsilon$ is closed: either $T_\emptyset(A_0) = \{\epsilon\}$ or $T_\varepsilon(A_0) = \{\epsilon\}$. Let us choose

$$\mathcal{R}_1 = \mathrm{Id}_{X^*} \cap X^{\leq 1} \times X^{\leq 1}. \tag{10.11}$$

If we note $A_0 = (\pi, S_0^-, S_0^+)$, then

$$[\pi, S_0^-, S_0^+, \mathcal{R}_1] = \{(\pi, S_0^-, S_0^+)\} \cup \{(\pi + 1, S_0^- \odot x, S_0^+ \odot x) \mid x \in X\},$$

where, $\forall x \in X$, $S_0^- \odot x \equiv S_0^+ \odot x \equiv \emptyset$. Using rule (R$'$3), we see that

$$[\pi, S, S', \mathcal{R}_1] \subseteq \mathrm{Cong}(\emptyset) \subseteq \mathrm{Cong}(\mathrm{im}(t_n)). \tag{10.12}$$

***$n > 0$, $\pi = 0$:*** Let $x$ be some node of $t_n$ such that $\exists S, S'$, $t_n(x) = (\pi, S, S')$ and $\pi < \bar{\pi}_n$. Let us denote by $W_x$ the word labeling the path from the root of $t_n$ (included) to $x$ (included).

*Case* 1. *$\exists x' \in \mathrm{dom}(t_n)$, $x'$ internal node, such that $t_n(x') = t_n(x)$.* As $\pi = 0$, the sons $x' \cdot 1, x' \cdot 2, \ldots, x' \cdot m$ of $x'$ are such that $t_n(x' \cdot 1) \cdot t_n(x' \cdot 2) \cdots t_n(x' \cdot m) \in T_A^{(\mathcal{O})}(W_{x'})$ for some oracle $\mathcal{O}$. Let us choose

$$\mathcal{R}_1 = \mathcal{O}(S, S') \cap X^{\leq 1} \times X^{\leq 1}. \tag{10.13}$$

Then

$$[\pi, S, S', \mathcal{R}_1] \subseteq \mathrm{im}(t_n). \tag{10.14}$$

*Case* 2. *$T_\emptyset(W_x) = \{\epsilon\}$ or $T_\varepsilon(W_x) = \{\epsilon\}$.* In this case the choice $\mathcal{R}_1 = \mathrm{Id}_{X^*} \cap X^{\leq 1} \times X^{\leq 1}$ again satisfies (10.12).

***$\pi > 0$:*** Let $x$ fulfill hypothesis (10.8). As $t_n$ is a proof-tree for $\mathcal{S}_{ABC}$, and as we suppose $x$ is closed and minimal (10.10), one of the following cases must occur.

*Case* 1. *$T_{cut}$ applies to $x$.* There exists $x' \in \mathrm{dom}(t_n)$, $\exists \pi' \in \mathbb{N}$, such that

$$t_n(x') = (\pi', S, S') \quad \text{and} \quad \pi' < \pi.$$

By the induction hypothesis

$$\exists \mathcal{R}_1 \in \mathcal{B}_1(S, S'), \quad [\pi', S, S', \mathcal{R}_1] \subseteq \mathrm{Cong}(\mathrm{im}(t_n)),$$

and by means of rule (R0),

$$[\pi, S, S', \mathcal{R}_1] \subseteq \mathrm{Cong}([\pi', S, S', \mathcal{R}_1]).$$

Hence (10.9) is true.

*Case* 2. *$T_\emptyset$ or $T_\epsilon$ applies to $x$.* Here again, the choice (10.11) fulfills property (10.12).

In the remaining cases we use the following notation: for every $k \in \mathbb{N}$ such that $t_k$ is not closed,

$$x_k = \min\{x \in \mathrm{dom}(t_k),\ x \text{ is not closed for } \mathcal{S}_{ABC} \text{ and } \exists S, S',\ t(x) = (\bar{\pi}_k, S, S')\}.$$

If $\exists k < n \mid t_k$ is not consistent or is closed, then by (7.10), $t_k = t_{k+1} = \cdots = t_n$; hence $\mathcal{R}(n, \pi) \Leftrightarrow \mathcal{R}(k, \pi)$, and this last property is true by the induction hypothesis.

Let us suppose now that $\forall k < n$, $t_k$ is consistent and unclosed. According to formula (7.9),

$$t_{k+1} = t_k[e_{k+1}/x_k]$$

for some tree of depth one, $e_{k+1}$.

Let $k \in [0, n-1]$, $x = x_k$, $\pi = \bar{\pi}_k$ (such a $k$ must exist because $x$ is internal). Let $x \cdot 1, \ldots, x \cdot \mu$ be the sequence of sons of $x$.

*Case* 3. *$T_A$ applies to $x$.* Hence there exists some oracle $\mathcal{O}$ such that $T_A^{(\mathcal{O})}$ applies to $x$. The choice (10.13) fulfills property (10.14).

*Case* 4. *$T_B^{\alpha}$ applies to $x$ (for some $\alpha \in \{-,+\}$).* Let us suppose $\alpha = +$. Let $x' = x(|x| - k_1)$ (the prefix of $x$ having length $|x| - k_1$), $t_n(x') = (\pi', \bar{U}, U')$. By definition of $\hat{\mathcal{S}}_{ABC}$, there exists some oracle $\mathcal{O}$ which is $\bar{\pi}_k$-consistent with $\mathrm{im}(t_k)$ and such that

$$\mu = 1 \quad \text{and} \quad t_n(x \cdot 1) = T_B^{(\mathcal{O}),+}(W_x).$$

Let us look at the proof of Lemma 7.2 in the particular case of this oracle $\mathcal{O}$: as the pairs $(u_\ell, u'_\ell)$ belong to $\mathcal{O}(\bar{U}, U')$ (for every $\ell \in [1, q]$) and $\pi' + |u_\ell| - 1 < \pi' + k_0 \leq \pi' + 2 \cdot k_0 < \bar{\pi}_k$, deduction (7.2) can be obtained just by using rules in $\mathcal{C}$. As deduction (7.2) is the only one (in the proof of Lemma 7.2) using rules in $\mathcal{B}_0 - \mathcal{C}$ we conclude that deduction (7.1) can be replaced by

$$\{t_n(x'), t_n(x \cdot 1)\} \cup \mathrm{im}(t_k) \overset{\langle * \rangle}{||\!\!-\!\!-}_{\mathcal{C}} \tau_{-1}(t_n(x)). \tag{10.15}$$

(We recall that $\tau_{-1}$ consists in replacing the weight of a given weighted equation into its predecessor.) Deduction (10.15) implies that

$$\exists p \in \mathbb{N}, \quad (\pi - 1, S, S') \in \mathrm{Cong}_p(\mathrm{im}(t_n)). \tag{10.16}$$

By the induction hypothesis, as $\pi - 1 < \bar{\pi}_n$, $\mathrm{im}(t_n)$ is locally $\pi - 1$-consistent, and hence, by Lemma 10.3, $\mathrm{im}(t_n)$ is $\pi - 1$-consistent. Hypothesis (10.16) implies that

$$\exists \mathcal{R}_1 \in \mathcal{B}_1(S, S'), \quad [\pi - 1, S, S', \mathcal{R}_1] \subseteq \mathrm{Cong}(\mathrm{im}(t_n)),$$

and hence, using (R0), that

$$\exists \mathcal{R}_1 \in \mathcal{B}_1(S, S'), \quad [\pi, S, S', \mathcal{R}_1] \subseteq \mathrm{Cong}(\mathrm{im}(t_n)).$$

*Case* 5. *$T_C$ applies to $x$.* By definition of $\hat{\mathcal{S}}_{ABC}$, there exists some oracle $\mathcal{O}$ which is $\bar{\pi}_k$-consistent with $\mathrm{im}(t_k)$ and such that

$$\mu = 1 \quad \text{and} \quad t_n(x \cdot 1) = T_C^{(\mathcal{O})}(W_x).$$

Let $W_x = A_1 \cdots A_\ell \cdots A_{|x|+1}$, $\kappa_1 < \cdots < \kappa_i < \kappa_{i+1} < \cdots \kappa_{D+1} = |x| + 1$, $\mathcal{S} = (\mathcal{E}_i)_{1 \leq i \leq D+1}$, where, for every $1 \leq i \leq d$,

$$\mathcal{E}_i = A_{\kappa_i} = \left( \pi_i, \sum_{j=1}^{d} \alpha_{i,j} S_j, \sum_{j=1}^{d} \beta_{i,j} S_j \right)$$

and

$$T_C^{(\mathcal{O})}(W_x) = \rho_e(\mathrm{INV}^{(\mathcal{O})}(\mathcal{S})), \quad \mathrm{W}^{(\mathcal{O})}(\mathcal{S}) \neq \perp, \quad \mathrm{D}^{(\mathcal{O})}(\mathcal{S}) = D \leq d - 1.$$

Let us look at the proof of Lemma 5.3 in the particular case of this oracle $\mathcal{O}$: the only place where a rule in $\mathcal{B}_0 - \mathcal{C}$ is used is in deduction (5.10), when Case 2, Subcase 1 (or Case 2, Subcase 2) of the recursive definition of $\mathrm{INV}^{(\mathcal{O})}(\mathcal{S})$ occurs. Let us recall that the pair $(u, u')$ chosen by the oracle $\mathcal{O}$ is such that

$$\mathcal{R} = \mathcal{O}\left(\sum_{j=1}^{d} \alpha_{1,j} S_j, \sum_{j=1}^{d} \beta_{1,j} S_j\right),$$

$$\nu = \mathrm{Div}(\alpha_{1,*}, \beta_{1,*}), \quad \mathcal{R}_\nu = \mathcal{R} \cap X^{\le \nu} \times X^{\le \nu}, \quad (u, u') \in \mathcal{R}_\nu.$$

Note that $\pi_1 + \nu - 1 < \pi_1 + 2 \cdot \nu < \pi_2 \le \mathrm{W}^{(\mathcal{O})}(\mathcal{S}) + 1 = \pi = \bar{\pi}_k$. As $\mathcal{O}$ is $\bar{\pi}_k$-consistent with $\mathrm{im}(t_k)$, we conclude that

$$\left(\pi_1 + |u|, \left(\sum_{j=1}^{d} \alpha_{i,j} S_j\right) \odot u, \left(\sum_{j=1}^{d} \beta_{i,j} S_j\right) \odot u'\right) \in \left[\pi_1, \sum_{j=1}^{d} \alpha_{i,j} S_j, \sum_{j=1}^{d} \beta_{i,j} S_j, \mathcal{R}_\nu\right] \subseteq \mathrm{Cong}(\mathrm{im}(t_k)).$$

Hence deduction (5.10) can be replaced by

$$\mathcal{E'}_1 \in \mathrm{Cong}(\mathrm{im}(t_k)). \tag{10.17}$$

Similarly, for every $i \in [2, D]$, as $\pi_i + 2 \cdot \mathrm{Div}(\alpha_{i,*}^{(i-1)}, \beta_{i,*}^{(i-1)}) < \pi_{i+1} \le \mathrm{W}^{(\mathcal{O})}(\mathcal{S}) + 1 = \pi = \bar{\pi}_k$, and $\mathcal{E}_i^{(i-1)} \in \mathrm{Cong}(\mathrm{im}(t_k))$,

$$(\mathcal{E}_i^{(i-1)})' \in \mathrm{Cong}(\mathrm{im}(t_k)). \tag{10.18}$$

It follows that deduction (5.9) can be replaced by

$$\{\mathrm{INV}^{(\mathcal{O})}(\mathcal{S})\} \cup \mathrm{im}(t_k) \overset{\langle * \rangle}{|\!\!\vdash\!\!-}_{\mathcal{C}} \tau_{-1}(t_n(x)). \tag{10.19}$$

Using the facts that $\rho_e(\mathrm{INV}^{(\mathcal{O})}(\mathcal{S})) \overset{\langle * \rangle}{|\!\!\vdash\!\!-}_{\mathcal{C}} \mathrm{INV}^{(\mathcal{O})}(\mathcal{S})$ and $\mathrm{im}(t_k) \subseteq \mathrm{im}(t_n)$ we may conclude that

$$\{t_n(x \cdot 1)\} \cup \mathrm{im}(t_n) \overset{\langle * \rangle}{|\!\!\vdash\!\!-}_{\mathcal{C}} \tau_{-1}(t_n(x)) = (\pi - 1, S, S'). \tag{10.20}$$

From (10.20) and the induction hypothesis, we can conclude, as in Case 4, that

$$\exists \mathcal{R}_1 \in \mathcal{B}_1(S, S'), \quad [\pi, S, S', \mathcal{R}_1] \subseteq \mathrm{Cong}(\mathrm{im}(t_n)).$$

(This ends the induction.)

By the above induction, for every $n \in \mathbb{N}$, $\mathrm{im}(t_n)$ is $\bar{\pi}_n$-consistent, i.e., $t_n$ is consistent. □

LEMMA 10.5. *$\hat{\mathcal{S}}_{ABC}$ is closed.*

*Proof.* Let $A_0 \in \mathcal{A}$. By Lemma 10.4, $\forall n \in \mathbb{N}$, $\hat{\mathcal{S}}_{ABC}^n(A_0)$ is consistent.

If $\hat{\mathcal{S}}_{ABC}^n(A_0)$ is consistent and is not closed, then, by definition (7.9),

$$\hat{\mathcal{S}}_{ABC}^n(A_0) \ne \hat{\mathcal{S}}_{ABC}^{n+1}(A_0);$$

if $\hat{\mathcal{S}}^n_{ABC}(A_0)$ is consistent and is closed, then, by definition (7.10),

$$\hat{\mathcal{S}}^n_{ABC}(A_0) = \hat{\mathcal{S}}^{n+1}_{ABC}(A_0).$$

Hence the equivalence (4.6), which defines the notion of closed global strategy, is fulfilled by $\hat{\mathcal{S}}_{ABC}$. □

THEOREM 10.6. *The formal systems $\mathcal{B}_0, \mathcal{B}_1$ are complete.*

*Proof.* By Lemma 9.1 $\hat{\mathcal{S}}_{ABC}$ is terminating on every unmarked assertion, and by Lemma 10.5 $\hat{\mathcal{S}}_{ABC}$ is closed. Let $A_0$ be some unmarked true assertion. According to the proof of Lemma 4.6, $\exists n_0 \in \mathbb{N}$ such that $t_\infty = \hat{\mathcal{S}}^{n_0}(A_0)$ is a proof-tree which is closed, and hence such that $\bar{\Pi}(t_\infty) = \infty$. By Lemma 10.5, $t_\infty$ is consistent, i.e., $\mathrm{im}(t_\infty)$ is $\infty$-consistent: $\forall(\pi, S, S') \in \mathrm{im}(t_\infty)$,

$$\exists \mathcal{R}_1 \in \mathcal{B}_1(S, S'), \quad [\pi, S, S', \mathcal{R}_1] \subseteq \mathrm{Cong}(\mathrm{im}(t_\infty));$$

hence,

$$\mathrm{im}(t_\infty) \overset{\langle * \rangle}{||\!\!-\!\!-}_{\mathcal{C}} [\pi, S, S', \mathcal{R}_1] \mid\!\!-\!\!-_{R4} (\pi, S, S'). \tag{10.21}$$

As the rules of $\mathcal{C}$ and (R4) are rules of $\mathcal{B}_1$, deduction (10.21) shows that

$$\mathrm{im}(t_\infty) \mid\!\!-\!\!-_{\mathcal{B}_1} (\pi, S, S'); \tag{10.22}$$

i.e., $\mathrm{im}(t_\infty)$ is a $\mathcal{B}_1$-proof.

In the general case where $A_0 = (\pi_0, U_0^-, U_0^+)$ might be marked, we observe that, owing to rules (R1), (R2), (R$'$3),

$$\{\rho_e(A_0)\} \overset{\langle * \rangle}{||\!\!-\!\!-}_{\mathcal{C}} \{A_0\}.$$

This deduction, combined with some $\mathcal{B}_1$-proof of $\rho_e(A_0)$, gives a $\mathcal{B}_1$-proof of $A_0$. □

THEOREM 10.7. *The bisimulation problem for rooted equational* 1*-graphs of finite out-degree is decidable.*

*Proof.* Let us consider the sequence of statements: Lemma 2.7, Lemma 2.8, Corollary 2.6 and Lemma 3.25. By means of the above statements, the bisimulation problem for rooted equational 1-graphs of finite out-degree reduces to the following decision problem (we call it the bisimulation problem for deterministic vectors):

*Instance:* a birooted, normalized dpda $\mathcal{M}$, its terminal alphabet $X$, a surjective literal morphism $\psi : X^* \to Y^*$ (we denote its kernel by $\bar{\psi}$), and $\lambda \in \mathbb{N} - \{0\}$, $S, S' \in \mathsf{DRB}_{1,\lambda}\langle\langle V \rangle\rangle$ (where $V$ is the structured alphabet associated with $\mathcal{M}$).

*Question:* $S \sim S'$? (where $\sim$ is the $\bar{\psi}$-bisimulation relation).

Let us consider $\mathcal{M}, X, V, \bar{\psi}$ given by some instance.

The equivalence relation $\sim$ on $\mathsf{DRB}_{1,\lambda}\langle\langle V \rangle\rangle$ has a recursively enumerable complement (this is well known). By Theorem 10.6 and Lemma 4.2, relation $\sim$ is recursively enumerable too. Hence $\sim$ is recursive.

But the function associating to every $\mathcal{M}, X, V, \bar{\psi}$ the corresponding deduction system $\mathcal{B}_1$ is recursive. Hence the bisimulation problem for deterministic vectors is decidable. □

## 10.2. System $\mathcal{B}_2$.

We exhibit here a deduction system $\mathcal{B}_2$ which is simpler than $\mathcal{B}_1$ and is still complete.

**Elementary rules.** Let us *eliminate* the weights in the rules of $\mathcal{B}_1$: we define a new set of assertions, $\mathcal{A}_2$, by

$$\mathcal{A}_2 = \bigcup_{\lambda \in \mathbb{N}-\{0\}} \mathsf{DRB}_{1,\lambda}\langle\langle\, V \,\rangle\rangle \times \mathsf{DRB}_{1,\lambda}\langle\langle\, V \,\rangle\rangle.$$

We define a binary relation $||\!\!\!-\!\!\!- \subseteq \mathcal{P}_f(\mathcal{A}_2) \times \mathcal{A}_2$, the *elementary deduction relation*, as the set of all the pairs having one of the following forms:

(R21)

$$\{(S,T)\} \,||\!\!\!-\!\!\!-\, (T,S)$$

for $\lambda \in \mathbb{N} - \{0\}$, $S,T \in \mathsf{DRB}_{1,\lambda}\langle\langle\, V \,\rangle\rangle$;

(R22)

$$\{(S,S'),(S',S'')\} \,||\!\!\!-\!\!\!-\, (S,S'')$$

for $\lambda \in \mathbb{N} - \{0\}$, $S,T \in \mathsf{DRB}_{1,\lambda}\langle\langle\, V \,\rangle\rangle$;

(R23)

$$\emptyset \,||\!\!\!-\!\!\!-\, (S,S)$$

for $S \in \mathsf{DRB}_{1,\lambda}\langle\langle\, V \,\rangle\rangle$;

(R′23)

$$\emptyset \,||\!\!\!-\!\!\!-\, (S,\rho_e(S))$$

for $S \in \mathsf{DRB}_{1,\lambda}\langle\langle\, V \,\rangle\rangle$;

(R24)

$$\{(S \odot x, T \odot x') \mid (x,x') \in \mathcal{R}_1\} \,||\!\!\!-\!\!\!-\, (S,T)$$

for $\lambda \in \mathbb{N} - \{0\}$, $S,T \in \mathsf{DRB}_{1,\lambda}\langle\langle\, V \,\rangle\rangle$, $(S \not\equiv \epsilon \wedge T \not\equiv \epsilon)$, and $\mathcal{R}_1 \in \bar{\mathcal{B}}_1$;

(R25)

$$\{(S_1 \cdot T + S, T)\} \,||\!\!\!-\!\!\!-\, (S_1^* \cdot S, T)$$

for $\lambda \in \mathbb{N} - \{0\}$, $S_1 \in \mathsf{DRB}_{1,1}\langle\langle\, V \,\rangle\rangle$, $S_1 \not\equiv \epsilon$, $(S_1,S) \in \mathsf{DRB}_{1,\lambda+1}\langle\langle\, V \,\rangle\rangle$, $T \in \mathsf{DRB}_{1,\lambda}\langle\langle\, V \,\rangle\rangle$;

(R26)

$$\{(S,S')\} \,||\!\!\!-\!\!\!-\, (S \cdot T, S' \cdot T)$$

for $\delta,\lambda \in \mathbb{N} - \{0\}$, $S,S' \in \mathsf{DRB}_{1,\delta}\langle\langle\, V \,\rangle\rangle$, $T \in \mathsf{DRB}_{\delta,\lambda}\langle\langle\, V \,\rangle\rangle$;

(R27)

$$\{(T_{i,*},T'_{i,*}) \mid 1 \le i \le \delta\} \,||\!\!\!-\!\!\!-\, (S \cdot T, S \cdot T')$$

for $\delta,\lambda \in \mathbb{N} - \{0\}$, $S \in \mathsf{DRB}_{1,\delta}\langle\langle\, V \,\rangle\rangle$, $T,T' \in \mathsf{DRB}_{\delta,\lambda}\langle\langle\, V \,\rangle\rangle$.

We define $|\!\!-\!\!-_{\mathcal{B}_2}$ as follows: for every $P \in \mathcal{P}_f(\mathcal{A}_2)$, $A \in \mathcal{A}_2$,

$$P \,|\!\!-\!\!-\, A \iff P \,\overset{\langle * \rangle}{||\!\!\!-\!\!\!-}\, \circ \,\overset{[1]}{||\!\!\!-\!\!\!-}_{23,24} \circ \,\overset{\langle * \rangle}{||\!\!\!-\!\!\!-}\, \{A\},$$

where $||\!\!\!-\!\!\!-_{23,24}$ is the relation defined by (R23), (R′23), (R24) only.

We define a simpler cost function $H_2 : \mathcal{A}_2 \to \mathbb{N} \cup \{\infty\}$ by

$$\forall (S, S') \in \mathcal{A}_2, \quad H_2(S, S') = \mathrm{Div}(S, S').$$

We let

$$\mathcal{B}_2 = \langle \mathcal{A}_2, H_2, \mid\!\!\!-\!\!-_{\mathcal{B}_2} \rangle.$$

LEMMA 10.8. *$\mathcal{B}_2$ is a deduction system.*

**Completeness.** Let us denote by $\mathcal{C}_2$ the subset of rules of $\mathcal{B}_2$ obtained by removing the weights in the rules of $\mathcal{C}$.

DEFINITION 10.9. *Let $P \in \mathcal{P}_f(\mathcal{A}_2)$. $P$ is said to be* self-generating *iff, for every $(S, S') \in P$,*

1. *either $S = S' = \epsilon$, or*
2. *$\exists \mathcal{R}_1 \in \bar{\mathcal{B}}_1(S, S')$, $\forall (x, x') \in \mathcal{R}_1$, $P \overset{\langle * \rangle}{|\!\!\mid\!\!\!-\!\!-}_{\mathcal{C}_2} (S \odot x, S' \odot x')$.*

(See Remark 10.12 below for the origins of this notion.)

LEMMA 10.10. *Let $A \in \mathcal{A}_2$ such that $A$ is unmarked. Then $H(A) = \infty$ iff there exists a finite self-generating set $P \subseteq \mathcal{A}_2$ such that $A \in P$.*

*Proof.* Owing to metarules (R23), (R24) it is clear that every self-generating set $P \in \mathcal{P}_f(\mathcal{A}_2)$ is a $\mathcal{B}_2$-proof. Hence, if $A$ belongs to some self-generating set, then $H(A) = \infty$.

Let us suppose now that $H_2(A) = \infty$. Let us consider the closed proof-tree $t_\infty$ obtained by applying the global strategy $\hat{S}_{ABC}$ on the assertion $(0, A)$. By Lemma 9.1 $t_\infty$ is finite, and by Lemma 10.5 $t_\infty$ is consistent, which means that $\mathrm{im}(t_\infty)$ is $\infty$-consistent. Let

$$P = pr_{2,3}(\mathrm{im}(t_\infty))$$

(where $pr_{2,3} : \mathcal{A} \to \mathcal{A}_2$ is the map erasing the weights).

As $\mathrm{im}(t_\infty)$ is $\infty$-consistent, $P$ is self-generating and $A \in P$. □

THEOREM 10.11. *$\mathcal{B}_2$ is a complete deduction system.*

*Proof.* We already noticed that every self-generating set is a $\mathcal{B}_2$-proof. Hence Lemma 10.10 proves that every true, unmarked assertion possesses some finite $\mathcal{B}_2$-proof.

Let $A$ be any true assertion. $\rho_e(A)$ has a finite proof $P$. Owing to rules (R1), (R2), (R′3), $Q = P \cup \{A\}$ is a $\mathcal{B}_2$-proof of $A$. □

**10.3. System $\mathcal{B}_3$.** We exhibit here a deduction system $\mathcal{B}_3$ which is even simpler than $\mathcal{B}_2$ and is still complete. Let us consider $\mathcal{B}_3 = \langle \mathcal{A}_3, H_3, \mid\!\!\!-\!\!-_{\mathcal{B}_3} \rangle$, where

$$\mathcal{A}_3 = \bigcup_{\lambda \in \mathbb{N} - \{0\}} \mathsf{DRB}_{1,\lambda}\langle\langle\, V_0 \,\rangle\rangle \times \mathsf{DRB}_{1,\lambda}\langle\langle\, V_0 \,\rangle\rangle.$$

$H_3 = H_{2|\mathcal{A}_3}$ and $\mid\!\!\!-\!\!-_{\mathcal{B}_3}$ is defined below: the metarules of $\mathcal{B}_3$ are essentially those of $\mathcal{B}_2$, but restricted to the unmarked vectors.

(R31)

$$\{(S, T)\} \,|\!\!\mid\!\!\!-\!\!-\, (T, S)$$

for $\lambda \in \mathbb{N} - \{0\}$, $S, T \in \mathsf{DRB}_{1,\lambda}\langle\langle\, V_0 \,\rangle\rangle$;

(R32)

$$\{(S,S'),(S',S'')\} \mid\!\!\vdash\!- (S,S'')$$

for $\lambda \in \mathbb{N} - \{0\}$, $S,T \in \mathsf{DRB}_{1,\lambda}\langle\langle\, V_0 \,\rangle\rangle$;

(R33)

$$\emptyset \mid\!\!\vdash\!- (S,S)$$

for $S \in \mathsf{DRB}_{1,\lambda}\langle\langle\, V_0 \,\rangle\rangle$;

(R34)

$$\{(S \odot x, T \odot x') \mid (x,x') \in \mathcal{R}_1\} \mid\!\!\vdash\!- (S,T)$$

for $\lambda \in \mathbb{N} - \{0\}$, $S,T \in \mathsf{DRB}_{1,\lambda}\langle\langle\, V_0 \,\rangle\rangle$, $(S \not\equiv \epsilon \wedge T \not\equiv \epsilon)$, and $\mathcal{R}_1 \in \bar{\mathcal{B}}_1$;

(R35)

$$\{(S_1 \cdot T + S, T)\} \mid\!\!\vdash\!- (S_1^* \cdot S, T)$$

for $\lambda \in \mathbb{N} - \{0\}$, $S_1 \in \mathsf{DRB}_{1,1}\langle\langle\, V_0 \,\rangle\rangle$, $S_1 \not\equiv \epsilon$, $(S_1, S) \in \mathsf{DRB}_{1,\lambda+1}\langle\langle\, V_0 \,\rangle\rangle$, $T \in \mathsf{DRB}_{1,\lambda}\langle\langle\, V_0 \,\rangle\rangle$;

(R36)

$$\{(S,S')\} \mid\!\!\vdash\!- (S \cdot T, S' \cdot T)$$

for $\delta, \lambda \in \mathbb{N} - \{0\}$, $S,S' \in \mathsf{DRB}_{1,\delta}\langle\langle\, V_0 \,\rangle\rangle$, $T \in \mathsf{DRB}_{\delta,\lambda}\langle\langle\, V_0 \,\rangle\rangle$;

(R37)

$$\{(T_{i,*}, T'_{i,*}) \mid 1 \le i \le \delta\} \mid\!\!\vdash\!- (S \cdot T, S \cdot T')$$

for $\delta, \lambda \in \mathbb{N} - \{0\}$, $S \in \mathsf{DRB}_{1,\delta}\langle\langle\, V_0 \,\rangle\rangle$, $T, T' \in \mathsf{DRB}_{\delta,\lambda}\langle\langle\, V_0 \,\rangle\rangle$.

We then define $\vdash\!-_{\mathcal{B}_3}$ as follows: for every $P \in \mathcal{P}_f(\mathcal{A}_3)$, $A \in \mathcal{A}_3$,

$$P \vdash\!-_{\mathcal{B}_3} A \iff P \overset{\langle * \rangle}{\mid\!\!\vdash\!-}_{\mathcal{B}_3} \circ \overset{[1]}{\mid\!\!\vdash\!-}_{33,34} \circ \overset{\langle * \rangle}{\mid\!\!\vdash\!-}_{\mathcal{B}_3} \{A\},$$

where $\mid\!\!\vdash\!-_{33,34}$ is now the relation defined by (R33), (R34) only.

As $\vdash\!-_{\mathcal{B}_3} \subseteq \vdash\!-_{\mathcal{B}_2}$, $H_3 = H_2$, it is clear that $\mathcal{B}_3$ is a deduction system.

**Completeness.** Let us call $\mathcal{C}_3$ the intersection of the set of the rules of $\mathcal{C}$ with the set of the rules of $\mathcal{B}_3$ (it is also equal to the set of instances of (R31), (R32), (R33), (R35), (R36), (R37)). Let us now call $P \in \mathcal{P}_f(\mathcal{A}_3)$ a $\mathcal{C}_2$-*self-generating* set iff it fulfills Definition 10.9 and a *self-generating* set iff it fulfills Definition 10.9 but where $\mathcal{C}_2$ is replaced by $\mathcal{C}_3$.

REMARK 10.12.

1. *This notion of a "self-generating set (of pairs)" is a straightforward adaptation to our d-space of vectors of the notion of the "self-proving set of pairs" defined in* [10, p. 162] *for the magma $M(F \cup \Phi, V)$.*

2. *The notion of "self-bisimulation" (introduced in* [5] *and also used in* [18, 17]*) was also such an adaptation, but in the context of a monoid-structure. The notion we use in this work can be seen, as well, as a generalization of this notion of self-bisimulation: when every class in $V_0/\smile$ has just one element, the only "rational deterministic boolean series" over $V_0$ are the words; in this case the self-bisimulations are exactly the self-generating sets.*

LEMMA 10.13. *Let $A \in \mathcal{A}_3$. Then $H_3(A) = \infty$ iff there exists a finite self-generating set $P \subseteq \mathcal{A}_3$ such that $A \in P$.*

*Proof.* Owing to metarules (R33) and (R34), every self-generating set is a $\mathcal{B}_3$-proof.

Let $A \in \mathcal{A}_3$ such that $H_3(A) = \infty$. By Lemma 10.10, there exists some $\mathcal{C}_2$-self-generating set $P$ such that $A \in P$.

Let us consider $Q = \{\rho_e(B) \mid B \in P\}$.

One can check that $\rho_e$ maps the set of rules of $\mathcal{C}_2$ into the set of rules of $\mathcal{C}_3$. One can also check that $\rho_e$ and $\odot$ are commuting (i.e., $\rho_e(S \odot u) = \rho_e(S) \odot u$). Hence $Q$ is such that, for every $(S, S') \in Q$,

1. either $S = S' = \epsilon$, or
2. $\exists \mathcal{R}_1 \in \bar{\mathcal{B}}_1(S, S')$, $\forall (x, x') \in \mathcal{R}_1$, $Q \overset{\langle * \rangle}{|\!\!\vdash\!\!-}_{\mathcal{C}_3} (S \odot x, S' \odot x')$.

That is, $Q$ is self-generating. □

THEOREM 10.14. *$\mathcal{B}_3$ is a complete deduction system.*

*Proof.* Lemma 10.13 implies the completeness property. □

**10.4. System $\mathcal{B}_4$.** We exhibit here a formal system $\mathcal{B}_4$ whose elementary rules can be considered as more natural than those of $\mathcal{B}_3$: they just consist in the rules expressing the basic algebraic properties of the bisimulation equivalence $\sim$ augmented with the grammatical rules (i.e., the rules of grammar $G_0$). This system $\mathcal{B}_4$ is still complete. Below, we just sketch the completeness proof, which is just a slight modification of the above completeness proofs.

Let us consider the alphabet $V_4 = V_0 \cup X$. The equivalence relation $\smile$ on $V_0$ is extended to $V_4$ as follows: for every $v, v' \in V_4$, $v \sim v'$ iff [$v \in V$, $v' \in V$, and there are equivalent in the sense used before] or [$v \in X$, $v' \in X$]. (This equivalence is the one considered in [15].) The right-action $\odot$ is extended to $\mathsf{B}\langle\langle\ V_4\ \rangle\rangle \times X^*$ as follows: for every $x \in X$, $\beta \in V_4^*$, $x' \in X$,

$$(x \cdot \beta) \odot x' = (x \cdot \beta) \bullet x', \tag{10.23}$$

$$\mathcal{A}_4 = \bigcup_{\lambda \in \mathbb{N} - \{0\}} \mathsf{DRB}_{1,\lambda}\langle\langle\ V_4\ \rangle\rangle \times \mathsf{DRB}_{1,\lambda}\langle\langle\ V_4\ \rangle\rangle.$$

This extension of $\odot$ leads naturally to extensions of the relation $\sim$ and of the notion of divergence. The cost $H_4$ is still defined by

$$\forall (S, S') \in \mathcal{A}_4, \quad H_4(S, S') = \mathrm{Div}(S, S').$$

**Elementary rules.**

(R41)

$$\{(S, T)\} \,|\!\!\vdash\!\!-\, (T, S)$$

for $\lambda \in \mathbb{N} - \{0\}$, $S, T \in \mathsf{DRB}_{1,\lambda}\langle\langle\ V_4\ \rangle\rangle$;

(R42)

$$\{(S, S'), (S', S'')\} \,|\!\!\vdash\!\!-\, (S, S'')$$

for $\lambda \in \mathbb{N} - \{0\}$, $S, T \in \mathsf{DRB}_{1,\lambda}\langle\langle\ V_4\ \rangle\rangle$;

(R43)

$$\emptyset \,|\!\!\vdash\!\!-\, (S, S)$$

for $S \in \mathsf{DRB}_{1,\lambda}\langle\langle\ V_4\ \rangle\rangle$;

(R$'$44)

$$\emptyset \mid\!\!\vdash\!\!- ((E_1, \ldots, E_i, \ldots, E_\lambda), (P_1, \ldots, P_i, \ldots, P_\lambda))$$

for $\lambda \in \mathbb{N} - \{0\}$, $(E_i)_{1 \leq i \leq \lambda}$, bijective numbering of some class in $V/\smile$ and $P_i$ equal to the right-hand side of $E_i$ in grammar $G_0$;

(R$''$44)

$$\{(S_x, T_{x'}) \mid (x, x') \in \mathcal{R}_1\} \mid\!\!\vdash\!\!- \left(\sum_{x \in X} x \cdot S_x, \sum_{x \in X} x \cdot T_x\right)$$

for $\lambda \in \mathbb{N} - \{0\}$, $S_x, T_x \in \mathsf{DRB}_{1,\lambda}\langle\langle\, V_4 \,\rangle\rangle$, and $\mathcal{R}_1 \in \bar{\mathcal{B}}_1$;

(R45)

$$\{(S_1 \cdot T + S, T)\} \mid\!\!\vdash\!\!- (S_1^* \cdot S, T)$$

for $\lambda \in \mathbb{N} - \{0\}$, $S_1 \in \mathsf{DRB}_{1,1}\langle\langle\, V_4 \,\rangle\rangle$, $S_1 \not\equiv \epsilon$, $(S_1, S) \in \mathsf{DRB}_{1,\lambda+1}\langle\langle\, V_4 \,\rangle\rangle$, $T \in \mathsf{DRB}_{1,\lambda}\langle\langle\, V_4 \,\rangle\rangle$;

(R46)

$$\{(S, S')\} \mid\!\!\vdash\!\!- (S \cdot T, S' \cdot T)$$

for $\delta, \lambda \in \mathbb{N} - \{0\}$, $S, S' \in \mathsf{DRB}_{1,\delta}\langle\langle\, V_4 \,\rangle\rangle$, $T \in \mathsf{DRB}_{\delta,\lambda}\langle\langle\, V_4 \,\rangle\rangle$;

(R47)

$$\{(T_{i,*}, T'_{i,*}) \mid 1 \leq i \leq \delta\} \mid\!\!\vdash\!\!- (S \cdot T, S \cdot T')$$

for $\delta, \lambda \in \mathbb{N} - \{0\}$, $S \in \mathsf{DRB}_{1,\delta}\langle\langle\, V_4 \,\rangle\rangle$, $T, T' \in \mathsf{DRB}_{\delta,\lambda}\langle\langle\, V_4 \,\rangle\rangle$.

**Completeness.**

PROPOSITION 10.15. *$\mathcal{B}_4$ is a complete deduction system.*

*Sketch of proof.* Let $(S, S') \in \mathcal{A}_4$ such that $S \sim S'$. Lemma 10.13 could be proved for assertions in $\mathcal{A}_4$ in the same way that it has been proved for assertions in $\mathcal{A}_3$. Hence there exists some self-generating set $P$ containing $(S, S')$.

In order to prove that $P$ is a $\mathcal{B}_4$-proof, we just have to check that every instance of (R34) belongs to $\vdash\!\!-_{\mathcal{B}_4}$.

Let $T, T' \in \mathsf{DRB}_{1,\lambda}\langle\langle\, V_4 \,\rangle\rangle$ (for some $\lambda \geq 1$). Using metarule (R$''$44) we get

(10.24)
$$\{(T \odot x, T' \odot x') \mid (x, x') \in \mathcal{R}_1\} \vdash\!\!-_{\mathcal{B}_4} \left\{\left(\sum_{x \in X} x \cdot (T \odot x), \sum_{x \in X} x \cdot (T' \odot x)\right)\right\}.$$

Using metarule (R$'$44) (as well as other auxiliary rules) we get

$$\emptyset \vdash\!\!-_{\mathcal{B}_4} \left\{\left(T, \sum_{x \in X} x \cdot (T \odot x)\right)\right\}; \quad \emptyset \vdash\!\!-_{\mathcal{B}_4} \left\{\left(T', \sum_{x \in X} x \cdot (T' \odot x)\right)\right\}. \tag{10.25}$$

From deductions (10.24), (10.25) and metarules (R41), (R42) one derives

$$\{(T \odot x, T' \odot x') \mid (x, x') \in \mathcal{R}_1\} \vdash\!\!-_{\mathcal{B}_4} \{(T, T')\}. \qquad \square$$

## 11. Comparisons and perspectives.

**11.1. Old tools.** We have reused here the notions developed in [34]:

- the *deduction systems* (which were in turn inspired by [9]);
- the *deterministic boolean series* (which were in turn inspired by [16]);
- the *deterministic spaces* (which were elaborated around the Meitus notion of linear independence [23, 24]);
- the *analysis* of the proof-trees generated by a suitable strategy (which was somehow similar to the analysis of the parallel computations, interspersed with replacement moves, done in [49, 31, 28]).

Some simplifications of [34] found by Stirling [46] were taken into account in this proof too:

- the technical notion of the "N-stacking sequence" is replaced by the slightly simpler notion of the "B-stacking sequence" (see section 8.2);
- the analysis of section 8 uses a choice of "generating set" which is simpler than the choice given in [33, 37];
- a main simplification linked with this more clever choice is that we can restrict ourselves to the case of a *proper, reduced* strict-deterministic grammar (as is done in [46]), while in [33, 37] we could not assume this restriction.

**11.2. New tools.** We also have introduced new ideas:

1. the notion of *$\eta$-bisimulation* over deterministic row-vectors of boolean series (which, in some sense, translates the usual notion of bisimulation to the d-space of row-vectors of series);
2. the notion of *oracle*, which is a choice of bisimulation for every pair of bisimilar vectors; the notion of triangulation of systems of linear equations is now "parametrized" by such an oracle $\mathcal{O}$ (see section 5.2); the strategies are now parametrized by an oracle too (see section 7);
3. the *elimination* argument: roughly speaking, this argument shows that, in a proof-tree $t$, if we take into account not only the *branch* ending at a node $x$, but also the *whole* proof-tree, then the metarule (R5)

$$\{(p, S, S')\} \mid\!\vdash\!- (p+2, S \odot x, S' \odot x')$$

is not needed to show that $\mathrm{im}(t) \vdash\!- \{t(x)\}$ (see section 10.1); a nice (and unexpected) by-product of this elimination is that the *weights* can be removed from the equations (see sections 10.2, 10.3).

**11.3. Perspectives and related works.** In view of the above main result (and of other closely related results), many directions for future work naturally arise:

- Whether the bisimulation problem is decidable or not, for equational graphs of arbitrary out-degree, which was raised by [6], remains an open problem.
- The class of processes of type $-1$ was introduced in [45]: they correspond to the computation-graphs of pda with only decreasing $\varepsilon$-moves. Whether the bisimulation problem is decidable or not for such processes is a natural challenge: the present article gives hope that it is decidable, while the negative results from [44, 22] suggest it might be undecidable.
- Once we know that the bisimulation problem is decidable for equational 1-graphs, it is natural to ask what the intrinsic complexity of this problem is. It is proved in [48] that the equivalence problem for *deterministic pda* is primitive recursive. It is tempting to examine what techniques could be extracted from this work and adapted to the above complexity problem (we

discuss in section 11.4 below the difficulty of such adaptations). In [43] a general algebraic tool (which generalizes the "extension-theorem" from [48]), called the "subwords lemma," is introduced for deterministic dpda. It is also tempting to try to adapt this tool to nondeterministic pda (we discuss in section 11.4 below the difficulty of such an adaptation).

- The intrinsic complexity of the bisimulation problem for some subclasses of graphs would be interesting too: let us quote the *context-free graphs* and the computation-graphs of *finite-turn* pda.
- The equivalence problem for *deterministic pushdown transducers* from a free monoid into a free group (or a linear group) is shown decidable in [38]. We strongly believe that this result can be unified with the result proved here (Theorem 10.7) into a more general statement saying that "the bisimulation problem is decidable for nondeterministic pushdown transducers, with outputs in a linear group and with only deterministic, decreasing, $\varepsilon$-moves" (i.e., whose underlying pda fulfills the hypotheses of the present article). Of course the notion of bisimulation for transducers has to be defined carefully.
- Let us recall that, given two directed graphs $G, G'$, labeled over the same alphabet $X$, $G'$ is a *quotient* of $G$ iff there exists a functional bisimulation from $G$ to $G'$. It is proved in [41] that the quotient problem is decidable for deterministic equational 1-graphs. The same decision problem remains open for the equational 1-graphs of finite out-degree. It is open for the subclass of context-free graphs too.

**11.4. Language equivalence versus bisimulation.** Let us stress here some differences between the behavior of the equivalence $\equiv$ (on one hand) and the behavior of $\sim$ (on the other hand), w.r.t. important algebraic notions. These differences explain some otherwise "odd" aspects of the present article, as well as point to difficulties that must be overcome for reaching the above-mentioned perspectives. For illustrating these differences, we shall always refer to the following example.

EXAMPLE 11.1. *Let $G = \langle X, V, P \rangle$, where*

$$X = \{a_i \mid 1 \leq i \leq 6\} \cup \{b_i \mid 1 \leq i \leq 6\} \cup \{h, h', h''\},$$

$$V = \{A_i \mid 1 \leq i \leq 6\} \cup \{B_i \mid 1 \leq i \leq 6\} \cup \{H, H', H''\},$$

$$P = \{(A_i, a_i) \mid 1 \leq i \leq 6\} \cup \{(B_i, b_i) \mid 1 \leq i \leq 6\} \cup \{(H, h), (H', h'), (H'', h'')\}.$$

*We define the equivalence relation $\smile$ on $V$ as the coarsest one:*

$$\forall v \in V, \quad [v]_{\smile} = V.$$

*The grammar $G$ is strict-deterministic and admits the partition $\smile$. Let $Y = \{a, b, h, h, h''\}$ and let $\Psi : X^* \to Y^*$ be the strict-alphabetical homomorphism defined by*

$$\Psi(a_i) = a, \quad \Psi(b_i) = b, \quad \Psi(h) = h, \quad \Psi(h') = h', \quad \Psi(h'') = h''.$$

*We define $\eta$ as the kernel of $\Psi$:*

$$\eta = \{(w, w') \in X^* \mid \Psi(w) = \Psi(w')\}.$$

**Direct product.** We show here that, unlike for the equivalence $\equiv$, relation $\sim$ over vectors does not easily reduce to the same relation over series, because it is *not* *compatible with direct product.* This explains the need for some *row-vectors* in the assertions as well as for some *matricial* rules in the systems $\mathcal{B}_i$ for $1 \leq i \leq 3$.

More precisely, it can be shown that, for every $(S_1, S_2), (T_1, T_2) \in \mathsf{DB}_{1,2}\langle\langle\ V\ \rangle\rangle$,

$$[(S_1, S_2) \equiv (T_1, T_2)] \Leftrightarrow [S_1 \equiv T_1 \text{ and } S_2 \equiv T_2], \tag{11.1}$$

while for the chosen series (see below)

$$[(S_1, S_2) \not\sim (T_1, T_2)] \quad \text{and} \quad [S_1 \sim T_1 \text{ and } S_2 \sim T_2]. \tag{11.2}$$

Let us choose:

$$S_1 = A_1 A_2; \quad S_2 = A_1 A_3 A_3; \quad T_1 = A_4 A_6; \quad T_2 = A_5 A_6 A_6.$$

The following binary relation $\mathcal{R}_1$ (resp., $\mathcal{R}_2$) over $X^*$ is a word-$\eta$-bisimulation for $(S_1, T_1)$ (resp., $(S_2, T_2)$):

$$\begin{aligned}
\mathcal{R}_1 = {} & \{(\varepsilon, \varepsilon)\} \cup \{(a_1, a_4)\} \cup \{(a_4 u, a_1 u) \mid u \in X^*\} \cup \{(a_i u, a_i u) \mid i \in \{2,3,5,6\}, u \in X^*\} \\
& \cup \{(a_1 a_2 u, a_4 a_6 u) \mid u \in X^*\} \cup \{(a_1 a_6 u, a_4 a_2 u) \mid u \in X^*\} \\
& \cup \{(a_1 a_i u, a_4 a_i u) \mid i \in \{1,3,4,5\}, u \in X^*\}, \\
\mathcal{R}_2 = {} & \{(\varepsilon, \varepsilon)\} \cup \{(a_1, a_5\}) \cup \{(a_5 u, a_1 u) \mid u \in X^*\} \cup \{(a_i u, a_i u) \mid i \in \{2,3,4,6\}, u \in X^*\} \\
& \cup \{(a_1 a_3, a_5 a_6)\} \cup \{(a_1 a_6 u, a_5 a_3 u) \mid u \in X^*\} \cup \{a_1 a_i u, a_5 a_i u \mid i \in \{1,2,4,5\}, u \in X^*\} \\
& \cup \{(a_1 a_3 a_3 u, a_5 a_6 a_6 u) \mid u \in X^*\} \cup \{(a_1 a_3 a_6 u, a_5 a_6 a_3 u) \mid u \in X^*\} \\
& \cup \{(a_1 a_3 a_i u, a_5 a_6 a_i u) \mid i \in \{1,2,4,5\}, u \in X^*\}.
\end{aligned}$$

Let us now check that

$$S_1 + S_2 \not\sim T_1 + T_2. \tag{11.3}$$

Let us consider some binary relation $\mathcal{R} \subseteq X^* \times X^*$ and show that it cannot be a word-$\eta$-bisimulation for $(S_1 + S_2, T_1 + T_2)$.

If $\mathcal{R}$ does not possess a pair with first component $a_1$, then it does not fulfill the totality condition.

If $(a_1, a_i) \in \mathcal{R}$, with $i \notin \{4, 5\}$, as $(S_1 + S_2) \odot a_1 = A_2 + A_3 A_3$ while $(T_1 + T_2) \odot a_i = \emptyset$, then $\mathcal{R}$ cannot be a word-$\eta$-bisimulation.

If $(a_1, a_4) \in \mathcal{R}$, as $(T_1 + T_2) \odot a_4 = A_6$, and the set of lengths of the words generated from $A_2 + A_3 A_3$ is $\{1, 2\}$ while the set of lengths of the words generated from $A_6$ is $\{1\}$, again $\mathcal{R}$ cannot be a word-$\eta$-bisimulation.

If $(a_1, a_5) \in \mathcal{R}$, as $A_2 + A_3 A_3 \not\sim A_6 A_6$ (apply the same argument on the set of lengths), then $\mathcal{R}$ cannot be a word-$\eta$-bisimulation.

We have proved (11.3).

But the properties established in section 4.3, namely, the soundness of rule (R8), applied with $T = T'$ equal to the column vector with two lines with entry $\varepsilon$ show that we must then have

$$(S_1, S_2) \not\sim (T_1, T_2).$$

This ends the proof of (11.2). Let us notice that, of course, the implication from left to right, in (11.1), also holds for $\sim$. It is the implication from right to left which may fail for $\eta$-bisimulation.

**Extension theorem.** We show here that the "extension theorem," which is the main new tool introduced in [48], does not hold for $\eta$-bisimulation.

*The vectors.* Let us consider the deterministic vectors and matrices $S, T \in \mathsf{DB}_{2,1}\langle\langle V \rangle\rangle$, $A \in \mathsf{DB}_{2,2}\langle\langle V \rangle\rangle$, $\alpha, \beta \in \mathsf{DB}_{1,2}\langle\langle V \rangle\rangle$:

$$S = \begin{pmatrix} H \\ \varepsilon \end{pmatrix}, \quad T = \begin{pmatrix} H' \\ \varepsilon \end{pmatrix}, \quad A = \begin{pmatrix} H'' & 0 \\ 0 & H \end{pmatrix},$$

$$\alpha = (\alpha_1, \alpha_2), \quad \beta = (\beta_1, \beta_2) \quad \text{with}$$

$$\begin{aligned}
\alpha_1 &= A_1,\\
\alpha_2 &= A_2H + A_3H' + A_4H' + A_5H'' + A_6H''\\
&\quad + B_1H + B_2H + B_3H' + B_4H' + B_5H'' + B_6H'',\\
\beta_1 &= B_1,\\
\beta_2 &= B_2H + B_3H' + B_4H' + B_5H'' + B_6H''\\
&\quad + A_1H + A_2H + A_3H' + A_4H' + A_5H'' + A_6H''.
\end{aligned}$$

*The equations.* The following equations are true:

$$\alpha \cdot S \sim \beta \cdot S, \tag{11.4}$$

$$\alpha \cdot AS \sim \beta \cdot AS, \tag{11.5}$$

$$\alpha \cdot T \sim \beta \cdot T. \tag{11.6}$$

The reason for these three equivalences is that

$$\alpha_1 \cdot H + \alpha_2 \sim \beta_1 \cdot H + \beta_2$$

(this equation is even true for language equivalence),

$$\alpha_1 \cdot H' + \alpha_2 \sim \beta_1 \cdot H' + \beta_2 \quad \text{and} \quad \alpha_1 \cdot H'' + \alpha_2 \sim \beta_1 \cdot H'' + \beta_2;$$

these last two equivalences are due to the fact that the number of occurrences of some $A_iH$ (resp., $A_iH'$, $A_iH''$, $A_i0$, $B_iH$, $B_iH'$, $B_iH''$, $B_i0$) in a deterministic sum does not modify its class modulo $\sim$, provided that this number remains nonnull (or remains null). We assert now that

$$\alpha \cdot AT \not\sim \beta \cdot AT. \tag{11.7}$$

Let us prove inequality (11.7). It reduces to

$$\alpha_1 \cdot H''H' + \alpha_2 \cdot H \not\sim \beta_1 \cdot H''H' + \beta_2 \cdot H,$$

but there is no word-$\eta$-bisimulation of depth 3 for the pair above, since

$$(\alpha_1 \cdot H''H' + \alpha_2 \cdot H) \odot a_1h''h' = \varepsilon,$$

while there is no pair $(a_1h''h', w) \in \eta$ such that

$$(\beta_1 \cdot H''H' + \beta_2 \cdot H) \odot w = \varepsilon.$$

*The extension theorem.* The "extension theorem" stated in [48, Theorem 1, p. 828] asserts that when $\sim$ is taken as the language equivalence relation, in the case where $n = 1$, $k = 0$, $m = \infty$, the conjunction of (11.4), (11.5), (11.6) implies

$$\alpha \cdot AT \sim \beta \cdot AT. \tag{11.8}$$

The above example shows that this theorem no longer holds when the equivalence relation $\sim$ is the $\eta$-bisimulation relation. (The same example also holds for depth $m = 3$ by the proof above.)

**Subwords lemma.** We show here that the "subwords lemma," which is the main new tool introduced in [43], does not hold for $\eta$-bisimulation.

The "subwords lemma" stated in [43, pp. 478–489] asserts that when $\sim$ is taken as the language equivalence relation, in the case where $\lambda = 2$, $n = \infty$, the conjunction of

$$\begin{aligned} \alpha \cdot S &\sim \beta \cdot S, \\ \alpha \cdot AS &\sim \beta \cdot AS, \\ \alpha \cdot BS &\sim \beta \cdot BS \end{aligned} \tag{11.9}$$

implies

$$\alpha \cdot ABS \sim \beta \cdot ABS. \tag{11.10}$$

But if we take

$$B = \begin{pmatrix} H' & 0 \\ 0 & H \end{pmatrix}$$

in the above example, the three premises (11.9) are valid, while (11.10) is equivalent to

$$\alpha_1 \cdot H''H'H + \alpha_2 \cdot HH \sim \beta_1 \cdot H''H'H + \beta_2 \cdot HH.$$

As the equivalence $\sim$ is right-cancellative (by Corollary 4.10, point (C3)), this equation is equivalent to

$$\alpha_1 \cdot H''H' + \alpha_2 \cdot H \sim \beta_1 \cdot H''H' + \beta_2 \cdot H,$$

which has been shown nonvalid in section 11.4.

**Unifiers.** We show here that, unlike for the equivalence $\equiv$, there is no unique most general unifier (modulo $\sim$) for a given finite system of linear equations (modulo $\sim$). This explains the need for some *oracle* in the definition of the triangulation process and, consequently, in the definition of strategy $T_C$.

It can be deduced from the type of reasoning used in [34], or even in the earlier paper [24], that any equation of the form

$$\alpha S \equiv \beta S \tag{11.11}$$

(where $\alpha, \beta$ are deterministic row-vectors in $\mathsf{DB}_{1,\lambda}\langle\langle V \rangle\rangle$) has a single "most-general unifier" up to $\equiv$, i.e., there exists a single column vector $S \in \mathsf{DB}_{\lambda,1}\langle\langle V \cup \mathcal{U} \rangle\rangle$, fulfilling (11.11) and such that all the solutions of (11.11) are obtained from $S$ by substituting an arbitrary element of $\mathsf{DB}_{1,1}\langle\langle V \rangle\rangle$ to every variable in $\mathcal{U}$ (here $\mathcal{U}$ is a new alphabet of variables, disjoint from $V$ and with cardinality $\leq \lambda$).

We have exhibited such an equation above, and we have seen that it admits at least three minimal unifiers,

$$\begin{pmatrix} HU \\ U \end{pmatrix}, \quad \begin{pmatrix} H'U \\ U \end{pmatrix}, \quad \begin{pmatrix} H''U \\ U \end{pmatrix},$$

where $U \in \mathcal{U}$ is a free variable. We could modify the example in such a way that the number of minimal unifiers matches an arbitrary value $p$. We can conclude that, in general, a left-linear equation modulo $\sim$ admits a finite number of minimal unifiers, but not a single one. This fact makes more difficult the treatment of systems of equations modulo bisimulation as compared to systems of equations modulo language equivalence.

**Appendix.** Let us sketch here a proof of Theorem 2.5.

LEMMA A.1. *Let $\Gamma = (\Gamma_0, v_0)$ be the computation 1-graph $(\mathcal{C}(\mathcal{M}), v_{\mathcal{M}})$ of some normalized pda $\mathcal{M}$. Then $\Gamma$ is equational and has finite out-degree.*

*Proof.* Let $\mathcal{M} = \langle X, Z, Q, \delta, q_0, z_0, F\rangle$ be a normalized pda. Let us consider a new letter $e \notin X$ and build the real-time pda $\mathcal{M}_e = \langle X \cup \{e\}, Z, Q, \delta_e, q_0, z_0, F\rangle$ obtained by setting that, for every $x \in X$ and $q \in Q$, $z \in Z$,

$$\delta_e(qz, x) = \delta(qz, x); \quad \delta_e(qz, e) = \delta(qz, \epsilon).$$

By [27, Theorem 2.6, p. 62], the computation-graph $\mathcal{C}(\mathcal{M}_e)$ is context-free, and by [3, Theorem 6.3, p. 187] every context-free graph is equational. Hence $\mathcal{C}(\mathcal{M}_e)$ is equational. Let us remark that $\mathcal{C}(\mathcal{M})$ is obtained from this graph just by contracting all the edges labeled by $e$. Let us contract the edges labeled by $e$ in some system of equations $S_e$ defining $\mathcal{C}(\mathcal{M}_e)$: we obtain a system of equations $S$ defining $\mathcal{C}(\mathcal{M})$. ∎

We now use the notation of [13]. Given a system of graph equations $S = \langle u_i = H_i;\ i \in [1, n]\rangle$, by $\mathcal{G}(S, u_i)$ we denote the $i$th component of the canonical solution of $S$.

DEFINITION A.2. *Let $S = \langle u_i = H_i;\ i \in [1, n]\rangle$ be a system of graph equations. It is* standard *iff it fulfills the following conditions:*

(1) *for every $i \in [1, n]$ and every distinct integer $k, \ell \in [1, \tau(H_i)]$, the sources $src(H_i, k)$, $src(H_i, \ell)$ are distinct vertices of $H_i$;*
(2) *for every $i \in [1, n]$ and every hyperedge $h$ of $H_i$ which is labeled by some unknown, all the vertices of $h$ are distinct;*
(3) *for every $i \in [1, n]$, $k \in [1, \tau(u_i)]$, $\lambda \in \mathbb{N}$, if there exist $\lambda$ edges going out of $src(\mathcal{G}(S, u_i), k)$, inside the graph $\mathcal{G}(S, u_i)$, then there exist also $\lambda$ edges going out of $src(H_i, k)$, inside the graph $H_i$.*

LEMMA A.3. *Let $S = \langle u_i = H_i;\ i \in [1, n]\rangle$ be a system of graph equations where the unknown $u_1$ has type 1. One can compute from $S$ a* standard *system of graph equations $S' = \langle u'_i = H'_i;\ i \in [1, n']\rangle$ such that the canonical solution of $S'$ has a first component $\mathcal{G}(S', u'_1) = \mathcal{G}(S, u_1)$.*

*Proof.* From $S$ one can construct a first system $S_1$ which generates the same first component $\mathcal{G}(S_1, u_1) = \mathcal{G}(S, u_1)$ and such that restrictions (1) and (2) of the lemma are fulfilled: this follows from [13, Proposition 2.10, p. 209] (notice that the condition "separated" in this reference is exactly the conjunction (1) $\wedge$ (2)).

Let $S_1 = \langle v_i = K_i;\ i \in [1, m]\rangle$. Let us replace every right-hand side $K_i$ by a finite hypergraph $L_i$ obtained by unfolding the graph $K_i$, according to the rules $v_j \to K_j$, as many times as necessary in order that every source $src(K_i, k)$ gets as many outgoing edges in $L_i$ as in the "complete unfolded graph" $\mathcal{G}(S_1, v_i)$. The new system $S' = \langle v_i = L_i;\ i \in [1, m]\rangle$ still fulfills conditions (1) and (2), it also fulfills condition (3), and for every $i \in [1, m]$, $\mathcal{G}(S_1, v_i) = \mathcal{G}(S', v_i)$. Hence $S'$ satisfies the conclusion of the lemma. ∎

LEMMA A.4. *Let $\Gamma = (\Gamma_0, v_0)$ be a rooted 1-graph over $X$ which is the first component of the canonical solution of some standard system of graph equations. Then $\Gamma$ is isomorphic to the computation 1-graph $(\mathcal{C}(\mathcal{M}), v_{\mathcal{M}})$ of some normalized pda $\mathcal{M}$.*

*Sketch of proof.* Let $S = \langle u_i = H_i;\ i \in [1, n]\rangle$ be a standard system of graph equations such that $\Gamma = \mathcal{G}(S, u_1)$.

Let us define $\mathcal{M} = \langle X, Z, Q, \delta, q_0, z_0, F\rangle$ as follows. In every right-hand side $H_i$ we number bijectively all the unknown hyperedges, $\{h_{1,i}, \ldots, h_{j,i}, \ldots, h_{n_i,i}\}$, and all the vertices, $\{v_{1,i}, \ldots, v_{q,i}, \ldots, v_{N_i,i}\}$. Let us denote by $\beta(j,i)$ the label of $h_{j,i}$.

$$Z = \{[j,i] \mid 1 \le i \le n, 1 \le j \le n_i\} \cup \{[1,0]\}.$$

(We extend $\beta$ by defining $\beta(1,0) = 1$.)

Intuitively every symbol $[j,i]$ describes the situation of a vertex which belongs to a component which has been glued to the $j$th unknown hyperedge of $H_i$.

Let $Q = [1,N]$, where $N$ is the maximum number of vertices in the graphs $H_i$. Intuitively, the transitions of $\mathcal{M}$ starting from a mode $q[j,i]$ describe the edges starting from the $q$th vertex of $H_{\beta(j,i)}$. Let us define precisely the transitions starting from a mode $q[j,i]$:

*Case* 1. *$q$ is strictly larger than the number of vertices of $H_{\beta(j,i)}$*. Then there is no transition starting from $q[j,i]$.

*Case* 2. *The vertex number $q$ of $H_{\beta(j,i)}$ is a source of $H_{\beta(j,i)}$ and $i \neq 0$*. Then

$$q[j,i] \xrightarrow{\varepsilon} q',$$

where $q'$ is the number of the vertex of $H_i$ to which it is glued (it is some vertex of $h_{j,i}$).

*Case* 3. *The vertex number $q$ of $H_{\beta(j,i)}$ is not a source of $H_{\beta(j,i)}$ or $i = 0$*.

*Internal edges*. For every edge $(v_{q,\beta(j,i)}, x, v_{q',\beta(j,i)})$, we add the transition

$$q[j,i] \xrightarrow{x} q'[j,i].$$

*External edges*. Let $k = \beta(j,i)$. For every $\ell$ such that $v_{q,\beta(j,i)}$ is a vertex of $h_{\ell,k}$ and every edge $(v_{r,\beta(\ell,k)}, x, v_{q',\beta(\ell,k)})$ where the vertex $v_{r,\beta(\ell,k)}$ of $H_{\beta(\ell,k)}$ is glued to the vertex $v_{q,\beta(j,i)}$ by the rewriting rule $u_{\beta(\ell,k)} \to H_{\beta(\ell,k)}$, we add the transition

$$q[j,i] \xrightarrow{x} q'[\ell,k][j,i].$$

The starting configuration is $1[1,0]$ (i.e., $q_0 = 1$, $z_0 = [1,0]$).

This pda is normalized (this is easy to check) and has a computation graph whose isomorphism class is exactly $\mathcal{G}(S, u_1)$ (this would be much more tedious to prove formally). $\square$

Theorem 2.5 clearly follows from these three lemmas.

**Acknowledgments.** I thank O. Burkart, D. Caucal, P. Jancar, F. Moller, and C. Stirling for useful discussion and information about the subject treated in this work. This work has also benefited from the incisive questions, comments, and criticism of O. Burkart and C. Stirling about [34].

## REFERENCES

[1] J. Baeten, J. Bergstra, and J. Klop, *Decidability of bisimulation equivalence for processes generating context-free languages*, in Proceedings of PARLE 87, Lecture Notes in Comput. Sci. 259, Springer-Verlag, Berlin, 1987, pp. 94–111.

[2] M. Bauderon, *Infinite hypergraph* I. *Basic properties (fundamental study)*, Theoret. Comput. Sci., 82 (1991), pp. 177–214.

[3] M. Bauderon, *Infinite hypergraph* II. *Systems of recursive equations*, Theoret. Comput. Sci., 103 (1992), pp. 165–190.

[4] M. Boffa, *Une remarque sur les systèmes complets d'identités rationnelles*, RAIRO Inform. Théor. Appl., 24 (1990), pp. 419–423.

[5] D. Caucal, *Graphes canoniques des graphes algébriques*, RAIRO Inform. Théor. Appl., 24 (1990), pp. 339–352.

[6] D. Caucal, *Bisimulation of context-free grammars and of pushdown automata*, in Modal Logic and Process Algebra, CSLI Lecture Notes 53, CSLI Publications, Stanford, CA, 1995, pp. 85–106.

[7] S. Christensen, Y. Hirshfeld, and F. Moller, *Bisimulation is decidable for basic parallel processes*, in Proceedings of CONCUR'93, Lecture Notes in Comput. Sci. 715, Springer-Verlag, Berlin, 1993, pp. 143–157.

[8] S. Christensen, H. Hüttel, and C. Stirling, *Bisimulation equivalence is decidable for all context-free processes*, Inform. and Comput., 121 (1995), pp. 143–148.

[9] B. Courcelle, *An axiomatic approach to the Korenjac-Hopcroft algorithms*, Math. Systems Theory, 16 (1983), pp. 191–231.

[10] B. Courcelle, *Fundamental properties of infinite trees*, Theoret. Comput. Sci., 25 (1983), pp. 95–169.

[11] B. Courcelle, *The monadic second-order logic of graphs* II. *Infinite graphs of bounded width*, Math. Systems Theory, 21 (1989), pp. 187–221.

[12] B. Courcelle, *Graph rewriting: An algebraic and logic approach*, in Handbook of Theoretical Computer Science, Vol. B, J. van Leeuwen, ed., MIT Press, Cambridge, MA, 1990, pp. 193–242.

[13] B. Courcelle, *The monadic second-order logic of graphs* IV. *Definability properties of equational graphs*, Ann. Pure Appl. Logic, 49 (1990), pp. 193–255.

[14] J. Groote and H. Hüttel, *Undecidable equivalences for basic process algebra*, Inform. and Comput., 115 (1994), pp. 354–371.

[15] M. Harrison, *Introduction to Formal Language Theory*, Addison-Wesley, Reading, MA, 1978.

[16] M. Harrison, I. Havel, and A. Yehudai, *On equivalence of grammars through transformation trees*, Theoret. Comput. Sci., 9 (1979), pp. 173–205.

[17] Y. Hirshfeld, M. Jerrum, and F. Moller, *A polynomial algorithm for deciding equivalence of normed context-free processes*, in Proceedings of FOCS'94, IEEE, 1994, pp. 623–631.

[18] H. Hüttel and C. Stirling, *Actions speak louder than words: Proving bisimilarity for context-free processes*, in Proceedings of LICS'91, IEEE, 1991, pp. 376–385.

[19] P. Jancar, *Bisimulation is decidable for one-counter processes*, in Proceedings of ICALP 97, Springer-Verlag, Berlin, 1997, pp. 549–559.

[20] D. Kozen, *A completeness theorem for Kleene algebras and the algebra of regular events*, in Proceedings of LICS'91, IEEE, 1991, pp. 214–225.

[21] D. Krob, *Complete systems of $\mathcal{B}$-rational identities*, Theoret. Comput. Sci., 89 (1991), pp. 207–343.

[22] R. Mayr, *Undecidability of weak bisimulation equivalence for* 1*-counter processes*, in Proceedings of ICALP 03, Lecture Notes in Comput. Sci. 2719, Springer-Verlag, Berlin, 2003, pp. 570–583.

[23] Y. Meitus, *The equivalence problem for real-time strict deterministic pushdown automata*, Kibernetika, 5 (1989), pp. 14–25 (in Russian; English translation in Cybernetics and Systems Analysis).

[24] Y. Meitus, *Decidability of the equivalence problem for deterministic pushdown automata*, Kibernetika, 5 (1992), pp. 20–45 (in Russian; English translation in Cybernetics and Systems Analysis).

[25] R. Milner, *A complete inference system for a class of regular behaviours*, J. Comput. System Sci., 28 (1984), pp. 439–466.

[26] R. Milner, *Communication and Concurrency*, Prentice-Hall, New York, 1989.

[27] D. Muller and P. Schupp, *The theory of ends, pushdown automata and second-order logic*, Theoret. Comput. Sci., 37 (1985), pp. 51–75.

[28] M. Oyamaguchi, *The equivalence problem for real-time d.p.d.a's*, J. Assoc. Comput. Mach., 34 (1987), pp. 731–760.

[29] D. Park, *Concurrency and automata on infinite sequences*, in Theoretical Computer Science, Lecture Notes in Comput. Sci. 104, Springer-Verlag, Berlin, 1981, pp. 167–183.

[30] H. Rogers, *Theory of Recursive Functions and Effective Calculability*, Series in Higher Mathematics, McGraw-Hill, New York, 1967.

[31] V. Romanovskii, *Equivalence problem for real-time deterministic pushdown automata*, Kibernetika, 2 (1985), pp. 13–23.

[32] A. Salomaa, *Two complete axiom systems for the algebra of regular events*, J. Assoc. Comput. Mach., 13 (1966), pp. 158–169.

[33] G. Sénizergues, *$\Gamma(A) \sim \Gamma(B)$?*, tech. report, nr1183-97, LaBRI, Université Bordeaux I, Talence, France, 1997; available online from http://www.labri.u-bordeaux.fr/~ges/.

[34] G. Sénizergues, *$L(A) = L(B)$?*, tech. report, nr1161-97, LaBRI, Université Bordeaux I, Talence, France, 1997.

[35] G. Sénizergues, *$L(A) = L(B)$?*, in Proceedings of INFINITY 97, Electronic Notes in Theoretical Computer Science 9, Elsevier, Amsterdam, 1997, pp. 1–26; available online from http://www.elsevier.nl/locate/entcs/volume9.html.

[36] G. Sénizergues, *The equivalence problem for deterministic pushdown automata is decidable*, in Proceedings ICALP 97, Lecture Notes in Comput. Sci. 1256, Springer-Verlag, Berlin, 1997, pp. 671–681.

[37] G. Sénizergues, *Decidability of bisimulation equivalence for equational graphs of finite outdegree*, in Proceedings of FOCS'98, R. Motwani, ed., IEEE Computer Society Press, Los Alamitos, CA, 1998, pp. 120–129.

[38] G. Sénizergues, *$T(A) = T(B)$?*, in Proceedings of ICALP 99, Lecture Notes in Comput. Sci. 1644, Springer-Verlag, Berlin, 1999, pp. 665–675. Full proofs in *$T(A) = T(B)$?*, tech. report 1209-99, LaBRI, Université Bordeaux I, Talence, France, 1999, pp. 1–61.

[39] G. Sénizergues, *Complete formal systems for equivalence problems*, Theoret. Comput. Sci., 231 (2000), pp. 309–334.

[40] G. Sénizergues, *$L(A) = L(B)$? Decidability results from complete formal systems*, Theoret. Comput. Sci., 251 (2001), pp. 1–166.

[41] G. Sénizergues, *Some applications of the decidability of dpda's equivalence*, in Proceedings of MCU'01, Lecture Notes in Comput. Sci. 2055, Springer-Verlag, Berlin, 2001, pp. 114–132.

[42] G. Sénizergues, *$L(A) = L(B)$? A simplified decidability proof*, Theoret. Comput. Sci., 281 (2002), pp. 555–608.

[43] G. Sénizergues, *The equivalence problem for t-turn dpda is co-NP*, in Proceedings of ICALP 03, Lecture Notes in Comput. Sci. 2719, Springer-Verlag, Berlin, 2003, pp. 478–489.

[44] J. Srba, *Undecidability of weak bisimilarity for pushdown processes*, in Proceedings of CONCUR 02, Lecture Notes in Comput. Sci. 2380, Springer-Verlag, Berlin, 2002, pp. 579–593.

[45] C. Stirling, *Decidability of bisimulation equivalence for normed pushdown processes*, in Proceedings of CONCUR 96, Lecture Notes in Comput. Sci. 1119, Springer-Verlag, Berlin, 1996, pp. 217–232.

[46] C. Stirling, *Decidability of DPDA Equivalence*, tech. report, Edinburgh ECS-LFCS-99-411, 1999, pp. 1–25; available online from http://www.lfcs.inf.ed.ac.uk/reports/99/ECS-LFCS-99-411/.

[47] C. Stirling, *Decidability of DPDA equivalence*, Theoret. Comput. Sci., 255 (2001), pp. 1–31.

[48] C. Stirling, *Deciding DPDA equivalence is primitive recursive*, in Proceedings of ICALP 02, Lecture Notes in Comput. Sci. 2380, Springer-Verlag, Berlin, 2002, pp. 821–832.

[49] L. Valiant, *The equivalence problem for deterministic finite-turn pushdown automata*, Inform. and Control, 25 (1974), pp. 123–133.

SIAM J. COMPUT.
Vol. 34, No. 5, pp. 1107–1128

# LEARNING RANDOM LOG-DEPTH DECISION TREES UNDER UNIFORM DISTRIBUTION*

JEFFREY C. JACKSON† AND ROCCO A. SERVEDIO‡

**Abstract.** We consider three natural models of random logarithmic depth decision trees over Boolean variables. We give an efficient algorithm that for each of these models learns all but an inverse polynomial fraction of such trees using only uniformly distributed random examples from $\{0,1\}^n$. The learning algorithm constructs a decision tree as its hypothesis.

**Key words.** computational learning theory, decision trees, PAC learning

**AMS subject classifications.** 68Q32, 68T05

**DOI.** 10.1137/S0097539704444555

## 1. Introduction.

Decision trees are widely used to represent various forms of knowledge. The apparent ease with which humans can understand and work with decision trees has also made them a popular form of representation for knowledge obtained through heuristic machine learning algorithms (see, e.g., [4, 10]). While heuristic algorithms have proved reasonably successful for many applications, there is some reason to believe that arbitrary decision trees are not efficiently learnable from random examples alone, as the class of decision trees is provably not efficiently learnable in the statistical query model, even when the examples are uniformly distributed [3].

Given the apparent difficulty of learning decision trees in polynomial time, many researchers have considered alternative learning scenarios. One line of work which has been pursued is to consider algorithms that run in superpolynomial time. Ehrenfeucht and Haussler [6] have shown that the class of size-$s$ decision trees over $\{0,1\}^n$ can be PAC learned in $n^{\log s}$ time steps. This result was later simplified by Blum [1]. Another approach which has been pursued is to study decision tree learnability in alternate learning models in which the learner has more power and the classifier produced need not be a decision tree itself. Kushilevitz and Mansour [9] gave a polynomial time algorithm which uses membership queries and can learn decision trees under the uniform distribution. The hypothesis produced by this algorithm is a weighted threshold of parity functions. Using different techniques, Bshouty [5] gave a polynomial time algorithm which learns decision trees in the model of exact learning from membership and (nonproper) equivalence queries; this implies that decision trees can be PAC learned in polynomial time under any distribution if membership queries are allowed. The hypothesis in this case is a depth-three Boolean circuit.

*Received by the editors June 18, 2004; accepted for publication (in revised form) November 30, 2004; published electronically June 3, 2005. This paper appeared in a preliminary form in a conference proceedings [8].
http://www.siam.org/journals/sicomp/34-5/44455.html

†Department of Mathematics and Computer Science, Duquesne University, Pittsburgh, PA 15282 (jackson@mathcs.duq.edu).

‡Department of Computer Science, Columbia University, 1214 Amsterdam Avenue, Mailcode 0401, New York, NY 10027-7003 (rocco@cs.columbia.edu). This author's research was partially supported by NSF Early Career Development (CAREER) grant CCF-0347282. This material is based upon work supported by the National Science Foundation under grant CCR-0209064.

**1.1. Our approach and results.** In this work we propose a third approach to coping with the difficulty of learning decision trees: looking at the average case. Since we have been unable to design algorithms which can learn all decision trees, we focus instead on algorithms which can efficiently learn most decision trees. Also, unlike some of the earlier approaches, our hypothesis is a decision tree.

We consider three natural models of random log-depth decision trees, i.e., decision trees over $n$ Boolean variables which are of depth $O(\log n)$. (Note that decision trees of logarithmic depth are a natural class to study in the context of uniform distribution learning, since under the uniform distribution any decision tree of poly($n$) size can be approximated to any inverse polynomial accuracy by a decision tree of $O(\log n)$ depth.) Our main result is a polynomial time algorithm that for each of these models learns, using decision tree hypotheses, all but a $1/p(n)$ fraction of such trees using only uniformly distributed random examples, where $p(n)$ is any polynomial function of $n$.

There are several motivations for this study. One natural motivation is that since decision tree learning is an interesting and important problem, yet worst-case theoretical analyses seem quite hard, it is natural to consider the average case. Another motivation comes from the work of Blum et al. [2], who proposed an approach to constructing cryptographic primitives such as pseudorandom generators and one-way functions based on the (presumed) intractability of certain learning problems. They defined a framework of learning from uniformly random examples in which the target function is also selected at random from the concept class according to some distribution. One of the main results of [2] is a proof that the existence of concept classes which are hard to learn in this average-case sense implies the existence of corresponding one-way functions whose circuit complexity is closely related to the circuit complexity of the concepts in the hard-to-learn class. Since decision trees are "simple" in terms of circuit complexity yet widely viewed as an intractable learning problem, one natural application of the Blum et al. paradigm would be to construct one-way functions based on the presumed intractability of decision tree learning. However, our results (which show that log-depth decision trees are *not* hard to learn in the average case) indicate that this approach does not yield secure one-way functions.

In section 2 we give the necessary background on uniform distribution learning and decision trees, and describe the three models of random decision trees which we consider. Section 3 gives useful Fourier properties of decision trees. In section 4 we present the learning algorithm, and sections 5 through 9 contain the proofs of correctness for the learning algorithm. We conclude in section 10.

## 2. Preliminaries.

A *Boolean decision tree* $T$ is a rooted binary tree in which each internal node has two children and is labeled with a variable, and each leaf is labeled with a bit $b \in \{-1, +1\}$. The children are ordered, i.e., each internal node has a definite left child and right child. We refer to an internal node whose two children are both leaves as a *preleaf node*. Because we will deal exclusively with Boolean decision trees in this paper, for convenience we will refer to them simply as decision trees.

A decision tree $T$ computes a Boolean function $f : \{-1,1\}^n \rightarrow \{-1,1\}$ in the obvious way: on input $x$, if variable $x_i$ is at the root of $T$, we go to either the left or right subtree depending on whether $x_i$ is $-1$ or 1. We continue in this fashion until reaching a bit leaf; the value of this bit is $f(x)$.

We define the *depth of a node* in a decision tree as follows. First, every decision tree must have at least one node; we do not admit the empty (0-node) decision tree. In a tree consisting of a single leaf node (labeled with some bit), the depth of this node is $-1$; we call such a tree *trivial.* The depth of the root in a nontrivial tree is 0, and the depth of any nonroot node is one greater than the depth of its parent. The *depth of a decision tree* $T$ is $-1$ if $T$ is trivial and the maximum depth over all preleaf nodes of $T$ otherwise.

A decision tree is *nonredundant* if no variable occurs more than once on any root-to-leaf path. We consider only nonredundant decision trees in this paper.

We let $\mathcal{U}$ be the uniform distribution on $\{-1,1\}^n$. We write $EX(f,\mathcal{U})$ to denote a uniform random example oracle for $f:\{-1,1\}^n \to \{-1,1\}$ which, when invoked, outputs a labeled example $\langle x, f(x)\rangle$ where $x$ is drawn from $\mathcal{U}$.

We say that a collection $\mathcal{C}$ of representations of Boolean functions (such as decision trees) is *PAC learnable under distribution* $\mathcal{D}$ *in* $t$ *time steps* if there is an algorithm $\mathcal{A}$ such that for every $f \in \mathcal{C}$ and for every positive $\epsilon$ and $\delta$, when $\mathcal{A}$ is run using $EX(f,\mathcal{D})$ as an oracle, then with probability at least $1-\delta$ (over the random draws from the oracle and any random choices made by $\mathcal{A}$) $\mathcal{A}$ runs for at most $t$ steps and produces a hypothesis function $h$ such that $\Pr_{a\sim\mathcal{D}}[f(a) \neq h(a)] \leq \epsilon$. We will focus on the specific question of PAC learnability of a subset of the class of all decision trees under the uniform distribution for $t$ bounded by a polynomial in $n$, $1/\epsilon$, and $1/\delta$. More generally, $\mathcal{C}$ is *PAC learnable in* $t$ *time steps* if for every distribution $\mathcal{D}$ it is PAC learnable under $\mathcal{D}$ in $t$ time steps.

We consider three models of random decision trees. Our primary model is the uniform distribution over the set of all nonredundant decision tree representations of depth at most $d$ on the variable set $\{x_1,\dots,x_n\}$. We call this the *uniform* model and will represent this distribution by $\mathcal{T}^U_{d,n}$. Note that not every Boolean function that can be represented by a depth-$d$ tree has the same probability mass under $\mathcal{T}^U_{d,n}$; some functions may have more $\mathcal{T}^U_{d,n}$-good trees which represent them than others. That is, $\mathcal{T}^U_{d,n}$ is a distribution over syntactic representations of decision tree functions, and not over the functions themselves.

In each of our other two models, the internal nodes form a complete tree of depth $d$ and are labeled uniformly at random using the variables $\{x_1,\dots,x_n\}$, with the restriction that the tree must be nonredundant. These models, denoted by $\mathcal{T}^C_{d,n}$ and $\mathcal{T}^B_{d,n}$, differ in that the leaves in $\mathcal{T}^C_{d,n}$ are selected independently and uniformly from $\{-1,1\}$, while in $\mathcal{T}^B_{d,n}$ each sibling pair must have opposite signs, although the sign of the left node is independently and uniformly chosen from $\{-1,1\}$. We call these the *complete* and *balanced* models, respectively.

We assume throughout that $d$ is $O(\log n)$ and that the learning algorithm knows the exact value of $d$. This latter assumption is without loss of generality (w.l.o.g.) since the algorithm can try all values $d=1,2,\dots$ until it succeeds.

**3. Fourier properties of decision trees.** We will be interested in carefully measuring the correlation between a decision tree $f$ and each of $f$'s variables. Define $e_i$ to be the $n$-bit vector that has a 1 in position $i$ and 0's elsewhere and define $\hat{f}(e_i)$ to be $E_{a\sim\mathcal{U}}[a_i f(a)]$. Since $f(a)$ and $a_i$ take values in $\{-1,1\}$ we have $\hat{f}(e_i) = \Pr_{a\sim\mathcal{U}}[f(a)=a_i] - \Pr_{a\sim\mathcal{U}}[f(a)\neq a_i]$. Each $\hat{f}(e_i)$ is a *first-order Fourier coefficient* of $f$.

Kushilevitz and Mansour [9] showed that decision trees have some particularly useful Fourier properties.[1] Define $L(i)$ to be the set of all leaves in a decision tree $f$

[1]In [9] Kushilevitz and Mansour considered decision trees in which internal nodes can contain arbitrary parity functions; however, as noted earlier we allow only single variables at internal nodes.

that are descendants of some node labeled by variable $x_i$, and let $d(\ell)$ represent the depth of a leaf $\ell$ in $f$. The analysis of [9] directly implies the following.

Corollary 3.1 (Kushilevitz–Mansour). *For every decision tree $f$ and every $1 \le i \le n$, there is a function $\sigma_i : L(i) \to \{-1, 1\}$ such that*

$$\hat{f}(e_i) = \sum_{\ell \in L(i)} 2^{-d(\ell)} \sigma_i(\ell).$$

Note that this corollary implies that in any tree of depth $d$, each $\hat{f}(e_i)$ is of the form $j/2^d$ for some integer $j$. This is because any leaf at depth $d+1$ must have a sibling leaf, so the total number of $\pm 1/2^{d+1}$ contributions to $\hat{f}(e_i)$ is even. We say that any first-order Fourier coefficient of the form $\frac{2k+1}{2^d}$ for some integer $k$ is an *odd coefficient*, and all other first-order coefficients are *even coefficients*.

From the above corollary we can obtain the following.

Lemma 3.2. *For every decision tree $f$ of depth $d$ and every $1 \le i \le n$, $\hat{f}(e_i)$ is an odd coefficient if and only if the total number of occurrences of the following conditions is odd for $x_i$:*

1. *A node at depth $d$ is labeled $x_i$ and the children of this node (both leaves) have opposite signs.*
2. *A leaf at depth $d$ has an ancestor labeled $x_i$.*
3. *A pair of sibling leaves at depth $d+1$ have the same sign, $x_i$ labels an ancestor of this pair of leaves, and $x_i$ is not the label of the parent of the pair.*

*Proof.* Fix an arbitrary $i$. By the corollary, only leaves in $L(i)$ that are at depth $d$ or $d+1$ can affect whether $\hat{f}(e_i)$ is an even or odd coefficient. Also, as noted earlier, each leaf at depth $d+1$ has a sibling leaf. Let "leaf set" denote either a leaf at depth $d$ or a sibling pair of leaves at depth $d+1$. Then notice that every leaf set in $L(i)$ is covered by at most one of the conditions of the lemma. We will show that each leaf set in $L(i)$ that is not covered by a condition contributes nothing to $\hat{f}(e_i)$, while every covered leaf set contributes $\pm 1/2^d$. From this, the lemma follows.

First, again by the corollary, each leaf in $L(i)$ at depth $d$ contributes $\pm 1/2^d$ to $\hat{f}(e_i)$. For sibling leaves at depth $d+1$ there are two cases: the siblings have either the same sign or different signs. If sibling leaves at depth $d+1$ have the same sign, then their immediate parent is irrelevant: the tree is equivalent to one with a leaf at depth $d$ in place of the parent node. Thus, in this case, the nonparent ancestor nodes each receive a $\pm 1/2^d$ contribution to their corresponding Fourier coefficients, while the parent node receives no contribution to its coefficient. For the case in which the siblings have different signs, it is not hard to see from the definition of $\hat{f}(e_i)$ that the net contribution to $\hat{f}(e_i)$ of the set of all bit-vectors $a$ that reach these two leaves will be $\pm 1/2^d$ if $x_i$ is the parent of the leaves and 0 if $x_i$ is a nonparent ancestor of the leaves. □

Conditions 2 and 3 of this lemma may seem redundant, since a tree with two sibling leaves having the same sign is equivalent to a tree that has a single leaf in place of the parent of the siblings. We include both conditions because these are syntactically different structures, both of which may arise in the various trees we consider.

A key observation which follows from this lemma is Lemma 3.3.

Lemma 3.3. *Fix any Boolean decision tree structure of depth $d$ and assign variables $x_1$ through $x_n$ arbitrarily to the internal nodes of the tree, with the constraint* *that the resulting tree is nonredundant. Assign each leaf bit by independently and uniformly selecting from* $\{-1,1\}$*. Then for every* $1 \le i \le n$ *such that* $x_i$ *is an ancestor of at least one leaf at depth* $d+1$*, we have* $\Pr[\hat{f}(e_i)$ *is an odd coefficient* $] = \frac{1}{2}$.

*Moreover, let* $S$ *be any subset of variables* $x_1, \ldots, x_n$ *with the following property: there is a collection* $C$ *of* $|S|$ *preleaves in* $T$*, each of which is at depth* $d$ *and is labeled with a different element of* $S$*, such that no variable in* $S$ *occurs on any of the paths from the root to any of these preleaves. Then we have that* $\Pr[$*for all* $i \in S$ $\hat{f}(e_i)$ *is an even coefficient* $] = \frac{1}{2^{|S|}}$.

*Proof.* We can view certain leaves of the tree as defining the "parity" of the internal nodes in a way that corresponds to the conditions of Lemma 3.2. More precisely, all internal nodes begin with even parity, and then their parities are computed by applying the following rules to each leaf and each pair of leaves (each leaf or leaf pair will meet the conditions of at most one rule):

1. If a pair of sibling leaves is at depth $d+1$ and the leaves have opposite signs, then the parity of their depth-$d$ parent node is toggled.
2. If a leaf is at depth $d$, then the parity of each ancestor node is toggled.
3. If a pair of sibling leaves is at depth $d+1$ and the leaves have identical signs, then the parity of each ancestor node except its parent node is toggled.

We can then define the parity of a variable $x_i$ as the parity of the parity of all nodes that are labeled $x_i$. It is clear that for each $i$, the first-order Fourier coefficient $\hat{f}(e_i)$ is odd if and only if $x_i$ has odd parity according to this definition.

Next, notice that since sibling leaves are labeled uniformly at random, any pair of sibling leaves at depth $d+1$ is equally likely to satisfy rule 1 or rule 3 above. So the probability that any given ancestor node of a pair of such sibling leaves has its parity toggled by a random assignment to these leaf bits is exactly 1/2. The same is true of the parity of the variable labeling the node. Since all leaves at depth $d+1$ have sibling leaves (because the tree depth is $d$), all possible assignments to leaves at depth $d+1$ are covered by the conditions of rules 1 and 3. Thus, any variable $x_i$ that is an ancestor of at least one leaf at depth $d+1$ will have odd parity with probability exactly 1/2, and the first equation is proved.

To see that the second equation also holds, observe that the $|S|$ preleaves in $C$ will each toggle the parity of the corresponding variable with probability 1/2. Since no variable in $S$ occurs on any path to a node in $C$, the final parities of these variables are all independent of each other, and the lemma follows. ∎

Kushilevitz and Mansour also observed that it follows from Corollary 3.1 and Chernoff bounds that for any decision tree $f$ and any $\delta > 0$, a uniformly distributed sample $S$ of $m$ labeled pairs $\langle x, f(x)\rangle$ is—with probability at least $1-\delta$ over the choice of $S$—sufficient to compute all of the first-order Fourier coefficients of $f$ exactly, for $m$ exponential in the depth $d$ of $f$ and polynomial in $\log(n/\delta)$. In particular, with probability at least $1-\delta$ over the random draw of $S$, $\hat{f}(e_i) = R\left((\sum_{a \in S} a_i f(a))/m\right)$, where $R(\cdot)$ represents "rounding" the argument to the nearest rational number having denominator $2^d$. Therefore, we also have the following corollary.

COROLLARY 3.4 (Kushilevitz–Mansour). *There is an algorithm FCExact such that, given* $\delta > 0$ *and access to* $EX(f,\mathcal{U})$ *for any decision tree* $f$ *of depth* $O(\log n)$*, with probability at least* $1-\delta$ *FCExact*$(n, \delta, EX(f,\mathcal{U}))$ *computes all of the first-order Fourier coefficients of* $f$ *exactly in time poly*$(n, \log(1/\delta))$.

For our algorithm we will need uniform random examples which are labeled not only according to the original tree $f$, but also according to certain subtrees obtained by restricting a subset of the variables of $f$. Each such subset will lie along a root-

```
**LearnTree**$(n, d, \delta, \epsilon, EX(T,\mathcal{U}))$
(is given number of variables $n$, the depth $d$ of the tree $T$ to be learned, desired
confidence $\delta > 0$, desired accuracy $\epsilon > 0$, and access to $EX(T,\mathcal{U})$).
 1.   if $d \le (\frac{1}{2}\log n)$
 2.     return UnikDTLearn$(n, \epsilon, \delta, EX(T,\mathcal{U}))$
 3.   endif
 4.   for $i = 1, \ldots, n$
 5.     for $b = -1, 1$
 6.       Call FCExact$(n-1, \delta/(4n^2), EX(T_{x_i \leftarrow b},\mathcal{U}))$ to compute
            $\widehat{T_{x_i \leftarrow b}}(e_j)$ for all $j \ne i$
 7.     endfor
 8.     if none of the coefficients $\widehat{T_{x_i \leftarrow b}}(e_j)$ is odd
 9.       return tree consisting of root $x_i$ and children defined by
            LearnTree$(n-1, d-1, \delta/4, \epsilon, EX(T_{x_i \leftarrow b}), \mathcal{U})$ for $b = -1, 1$.
10.     endif
11.   endfor
12.   return "fail"
```

FIG. 4.1. *The algorithm for learning random log-depth decision trees.*

to-leaf path in $f$ and—since we consider only trees of depth $O(\log n)$—will therefore have cardinality $O(\log n)$. We can simulate exactly an example oracle for such a restricted $f'$ given an oracle $EX(f,\mathcal{U})$ by simply drawing examples from $EX(f,\mathcal{U})$ until we obtain one that satisfies the restriction on the $O(\log n)$ variables. Since each example from $EX(f,\mathcal{U})$ will satisfy such a restriction with probability $1/\text{poly}(n)$, the probability of failing to obtain such an example after poly$(n)$ many draws from $EX(f,\mathcal{U})$ can be made exponentially small. Thus the simulation of these subtree oracles is both exact and efficient. We will use $EX(f',\mathcal{U})$ to represent the simulated oracle for a restriction $f'$.

## 4. The algorithm for learning random decision trees.

Our algorithm for learning random decision trees, which we call LearnTree (see Figure 4.1), operates in two phases. In the first phase (lines 4–11) the algorithm uses the Fourier properties outlined above to find a root-like variable for the original tree. (Informally, a root-like variable has the property that it can be taken as the root of the tree without increasing the depth of the tree; we give precise definitions later.) Once this is done, it recursively finds a good root for each of the two subtrees induced by this root, and so on. The process stops at depth $d - \frac{1}{2}\log n$, so when it stops there are at most $2^d/\sqrt{n}$ subtrees remaining, each of depth at most $\frac{1}{2}\log n$. (Recall that w.l.o.g. we may assume the algorithm knows the exact value of $d$.) In the second phase (lines 1–3) we employ an algorithm UnikDTLearn$(n, \epsilon, \delta, EX(T,\mathcal{U}))$ due to Hancock [7] to learn the remaining "shallow" decision trees. (If $d < \frac{1}{2}\log n$, then our algorithm performs only the second phase.)

The intuition underlying the algorithm is that at each step in the first phase, each of the two subtrees of the root $x_i$ of a decision tree $T$ will obviously have depth at least one less than that of the original tree. These subtrees will therefore contain no odd first-order Fourier coefficients by Lemma 3.2, and thus the root $x_i$ will pass the test at line 8. On the other hand, we will show that in our random decision tree models, if we consider a variable $x_j$ which is not the root (or, more accurately, is not root-like in a sense defined below), then projecting on $x_j$ will result in at least one projection containing odd first-order coefficients. (This will follow from our earlier Fourier analysis of decision trees plus some combinatorial arguments showing that if $x_j$ is not root-like, then with very high probability the trees resulting from restricting on $x_j$ will have an $\omega(\log n)$-size collection $C$ of preleaves as in Lemma 3.3.)

Hancock's UnikDTLearn is efficient, so LearnTree clearly runs in time poly$(n)$ for any value $d = O(\log n)$. The rest of this paper shows that the algorithm with high probability outputs an accurate decision tree for our various models of random decision trees.

## 5. Bottlenecks and recursing the algorithm.

The first phase of our algorithm attempts to select recursively the root of the original tree $T$ and its subtrees. One difficulty is that $T$ may have more than one variable that "acts like" a root, in the sense that putting any of these variables at the root does not increase the depth of the tree. For example, consider a decision tree representation of the function $x_1 \oplus x_2$. Although we might represent this with a tree $T$ having $x_1$ at the root, it can be represented equally well by a tree with $x_2$ at the root.

The following definition captures the notion of a "root-like" variable for us.

DEFINITION 5.1. *A variable $x_i$ is a* bottleneck *for a decision tree $T$ if $T$ is nontrivial and $x_i$ occurs on every root-to-leaf path in $T$.*

Clearly the variable labeling the root is always a bottleneck for any tree. We note that if $x_i$ is the root of a tree $T$, then a variable $x_j \neq x_i$ is a bottleneck in $T$ if and only if $x_j$ is a bottleneck in both the left and right subtrees of $T$. Notice also that any bottleneck variable will pass the test at line 8 of LearnTree, so some variable $x_i$ will pass the test at every stage of the recursion given that FCExact returns accurate values for all first-order Fourier coefficients.

For the rest of this section let $\mathcal{T}_{d,n}$ denote any fixed one of the three random tree models $\mathcal{T}^B_{d,n}, \mathcal{T}^C_{d,n}, \mathcal{T}^U_{d,n}$. In later sections we will show that for a random tree $T$ drawn from $\mathcal{T}_{d,n}$, any nonbottleneck variable will with very high probability not pass the test of line 8. Thus each recursive call of LearnTree is performed by restricting some bottleneck variable; however, the bottleneck may or may not be the root. If the root is the bottleneck chosen, then it is easy to see that each of the two subtrees will be drawn from $\mathcal{T}_{d-1,n-1}$ (over a suitable set of $n-1$ variables) as desired, and the inductive assumption of LearnTree (that the tree it is given is drawn from $\mathcal{T}_{d,n}$) will be valid. If a nonroot bottleneck is chosen, however, it is not a priori clear that the two resulting subfunctions for the recursive call correspond to draws from $\mathcal{T}_{d-1,n-1}$.

We will show shortly that as long as any bottleneck is chosen, the two resulting subfunctions do indeed correspond to draws from $\mathcal{T}_{d-1,n-1}$. (The correspondence is exact for the complete model; for the other models the subfunctions correspond to draws from a distribution which has negligible statistical difference from $\mathcal{T}_{d-1,n-1}$.) While the two draws are not independent of each other, this does not negatively impact the algorithm.

First, we define some terms. Figure 5.1 defines a tree restructuring operation we call a *root swap.* Notice that this operation can be performed on a tree only if the children of the root are both labeled with the same variable; we call a tree with this property *root swappable.* More generally, a *swap* operation takes a tree $T$ and a node $\eta$ in the tree such that the subtree $S$ rooted at $\eta$ is root swappable and returns a tree $T'$ which is identical to $T$ except that the subtree rooted at $\eta$ is replaced with the root swap of $S$.

We next define an equivalence relation on decision trees which we call *structural*

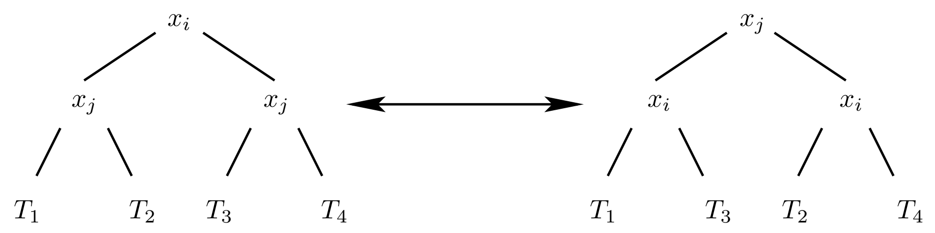


FIG. 5.1. *Applying a root swap operation to the tree on the left (right) produces the tree on the right (left). Note that the tree produced by a root swap computes the same function as the original tree.*

*equivalence*. Formally, this relation is defined inductively as follows.

DEFINITION 5.2. (i) *Two decision trees $T$ and $T'$, both of depth $d < 1$, are structurally equivalent if and only if they are identical.* (ii) *Two decision trees $T$ and $T'$, both of depth $d \geq 1$, are structurally equivalent if and only if there exists a sequence of swap operations that will transform $T'$ to a tree $T''$ such that $T$ and $T''$ have the same root variable and each of the child subtrees of the root of $T''$ is structurally equivalent to its corresponding subtree in $T$.* (iii) *Two decision trees $T$ and $T'$ of different depths are not structurally equivalent.*

Informally, two decision trees $T$ and $T'$ are structurally equivalent if $T'$ can be obtained from $T$ by performing a sequence of swaps. Note that if two trees are structurally equivalent, then they compute the same function. The following lemma, which we prove in Appendix A, shows that bottleneck variables can be swapped to the root of a tree in a way that preserves structural equivalence.

LEMMA 5.3. *Let $T$ be any decision tree. If variable $x_i$ is a bottleneck for $T$, then there is a tree $T'$ having $x_i$ at its root that is structurally equivalent to $T$.*

Let $\mathcal{T}^i_{d,n}$ be the induced distribution over trees obtained by restricting $\mathcal{T}^C_{d,n}$ to trees for which $x_i$ is a bottleneck, and let $\mathcal{T}^{\tilde{i}}_{d,n}$ be the distribution over trees obtained by first selecting a tree $T$ according to $\mathcal{T}^i_{d,n}$ and then performing a minimal sequence of swap operations (as implicitly described in the proof of Lemma 5.3) to produce a structurally equivalent $\tilde{T}$ having $x_i$ as its root. Finally, let $\mathcal{T}^{-1}_{d-1,n-1}$ (resp., $\mathcal{T}^{1}_{d-1,n-1}$) represent the distribution over trees corresponding to a random variable that selects a tree $\tilde{T}$ according to $\mathcal{T}^{\tilde{i}}_{d,n}$ and then returns the left (resp., right) subtree as the value of the random variable. Then for the complete model, it suffices to prove the following lemma.

LEMMA 5.4. *Let $\mathcal{T}^i_{d,n}$, $\mathcal{T}^{-1}_{d-1,n-1}$, and $\mathcal{T}^{1}_{d-1,n-1}$ be as defined above. Then for all $1 \leq i, d \leq n$, $\mathcal{T}^{-1}_{d-1,n-1}$ and $\mathcal{T}^{1}_{d-1,n-1}$ are both identical to $\mathcal{T}^{C}_{d-1,n-1}$.*

*Proof.* The proof is by induction on $d$. For the base case $d = 1$, any tree $T$ drawn from $\mathcal{T}^i_{1,n}$ has either $x_i$ at the root or some other variable $x_j$ at the root and $x_i$ as the root of both children of $x_j$. In either case, the corresponding tree $\tilde{T}$ of the process defining $\mathcal{T}^{\tilde{i}}_{1,n}$ will have $x_i$ at the root with two depth-0 children. It is easy to see that, over random draws from $\mathcal{T}^{\tilde{i}}_{1,n}$, the root variables of these children of $x_i$ are each uniformly distributed over the $n-1$ variables excluding $x_i$ (although the distributions of these root variables are not necessarily independent). The values of the leaves are also uniformly and independently distributed. Therefore, the base case has been shown.

For the inductive case, consider a tree $T$ drawn from $\mathcal{T}^i_{d,n}$ for fixed $d > 1$. Since $x_i$ must be a bottleneck in $T$, it either is the root of $T$ or is a bottleneck in both children of the root of $T$. If $x_i$ is the root of $T$, the lemma obviously holds. So we are left with the case in which some variable $x_j$—uniformly chosen from the $n-1$ variables excluding $x_i$—labels the root of $T$ and $x_i$ is a bottleneck in both children of $x_j$. Let $T_{-1}$ ($T_1$) represent the left (right) subtree of $T$, and let $\tilde{T}_{-1}$ ($\tilde{T}_1$) represent the tree obtained by swapping $x_i$ to the root of $T_{-1}$ ($T_1$). Since $x_i$ is a bottleneck in $T_{-1}$ ($T_1$), the children of $x_i$ in $\tilde{T}_{-1}$ ($\tilde{T}_1$) are drawn from $\mathcal{T}^C_{d-2,n-2}$, by the inductive hypothesis.[2] Notice also that, although the distribution over each child of $x_i$ in $\tilde{T}_{-1}$ may be dependent on its sibling's distribution, each is independent of both of the distributions over children of $x_i$ in $\tilde{T}_1$. Therefore, after performing a final swap of the $x_i$'s at the roots of $\tilde{T}_{-1}$ and $\tilde{T}_1$ with the root $x_j$ of $T$, we obtain a tree $\tilde{T}$ in which each child of the root $x_i$ is a tree rooted at uniformly (over $n-1$ variables) chosen $x_j$ and in which the children of $x_j$ are independently distributed according to $\mathcal{T}^C_{d-2,n-2}$. That is, each child of $x_i$ in $\tilde{T}$ is distributed according to $\mathcal{T}^C_{d-1,n-1}$. Since $\tilde{T}$ is by construction distributed according to $\mathcal{T}^{\tilde{i}}_{d,n}$, the lemma follows. $\square$

This proof does not immediately apply to either of the other two tree models. The problem for balanced trees is in the base case: a swap in a balanced tree of depth 1 can produce a tree that is no longer balanced. For uniform trees, there is a technical difficulty in that the distribution over children will only be exactly $\mathcal{T}^U_{d-1,n-1}$ in the case in which no swap is performed (the root is chosen as the bottleneck variable). If a nonroot bottleneck is chosen, a swap must be performed, and the children of the root then must not be leaves. But leaves have nonzero probability in $\mathcal{T}^U_{d-1,n-1}$, so the children are not distributed according to $\mathcal{T}^U_{d-1,n-1}$ in this case.

We can, however, still say that with very high probability over the choice of tree according to either $\mathcal{T}^B_{d,n}$ or $\mathcal{T}^U_{d,n}$, the choice of bottleneck variable on which to recurse will not affect the distribution over the recursive subtree problems. We use two lemmas to show this. First, we will later prove the following lemma in the context of some of our other combinatorial lemmas in Appendix E.

LEMMA 5.5. *Let $T$ be drawn from $\mathcal{T}_{d,n}$ where $d \geq \frac{1}{2}\log n$ and $d = O(\log n)$. The probability that $T$ has any bottleneck variable which occurs at some depth $k \geq \frac{1}{8}\log n$ is $1/n^{\omega(1)}$.*

For the balanced model, this means that the sequence of swaps performed to transform $T$ chosen according to $\mathcal{T}^B_{d,n}$ into $\tilde{T}$ having bottleneck $x_i$ at the root will almost certainly not perform a swap involving $x_i$ nodes whose children are leaf bits. And a simple modification to the proof of Lemma 5.4 shows the following.

LEMMA 5.6. *Let $\mathcal{T}^{\bar{i}}_{d,n}$ be the induced distribution over trees obtained by restricting $\mathcal{T}^B_{d,n}$ to trees for which $x_i$ is a bottleneck that does not appear at depth $d$. Let $\mathcal{T}^{-1}_{d-1,n-1}$ and $\mathcal{T}^1_{d-1,n-1}$ be induced distributions defined as before except relative to $\mathcal{T}^{\bar{i}}_{d,n}$ rather than $\mathcal{T}^i_{d,n}$. Then for all $1 \leq i, d \leq n$, $\mathcal{T}^{-1}_{d-1,n-1}$ and $\mathcal{T}^1_{d-1,n-1}$ are both identical to $\mathcal{T}^B_{d-1,n-1}$.*

So in the balanced model, there is a negligible chance that the choice of bottleneck variable will negatively affect the algorithm. For the uniform model, we will also use the following lemma, which is again proved in Appendix E.

LEMMA 5.7. *Let $T$ be drawn from $\mathcal{T}^U_{d,n}$, where $d \geq \frac{1}{4}\log n$. The probability that $T$ has a leaf at depth less than $d - \frac{1}{4}\log n$ is $1/2^{n^{\Omega(1)}}$.*

[2]Strictly speaking, the children of $x_i$ are fixed for any given $T$. What we are actually claiming here is that over draws of $T$ from $\mathcal{T}^i_{d,n}$, for fixed $d > 1$, the children of $x_i$ in $\tilde{T}_{-1}$ are distributed according to $\mathcal{T}^C_{d-2,n-2}$. But for ease of exposition, here and below we often blur the distinction between a single tree produced by one application of a random process defining a distribution over trees and the distribution itself.

Combining this with Lemma 5.5, for $d \geq \frac{1}{2}\log n$, with probability at least $1 - 1/n^{\omega(1)}$ (which we will refer to in this section as "very high probability"), a tree $T$ drawn from $\mathcal{T}^U_{d,n}$ has no leaves through depth $d - \frac{1}{4}\log n$, while all bottleneck variables are at shallower depths. Fix $t$ to be any depth ($\frac{3}{16}\log n$ will do) such that with very high probability all the bottleneck variables in $T$ drawn from $\mathcal{T}^U_{d,n}$ have depth less than $t$ and all leaves have depth greater than $t$.

Notice that (with very high probability) we have $2^t$ nodes at depth $t$ in $T$ and that the subtrees rooted at these nodes are independently and identically (up to variable renaming) distributed according to $\mathcal{T}^U_{d-t,n-t}$. Furthermore, notice that all of the nodes labeled by any bottleneck variable are with very high probability located in the portion of the tree above these new "leaves." So the proof of Lemma 5.4 shows that, with very high probability, swapping a bottleneck variable $x_i$ to the root of $T$ produces a tree $\tilde{T}$ in which the children of $x_i$ are each distributed according to the complete distribution down to depth $t-1$ and then according to $\mathcal{T}^U_{d-t,n-t}$. But this is just $\mathcal{T}^U_{d-1,n-1}$ restricted to contain no leaves until depth $t$, which by Lemma 5.7 is negligibly different from $\mathcal{T}^U_{d-1,n-1}$.

Therefore, in the uniform model, the recursive distributions obtained if a nonroot bottleneck is chosen are negligibly different from the distributions that would be obtained if the root variable were selected. Thus, in both the balanced and the uniform models, the probability that LearnTree fails as a result of distribution differences induced by selecting a nonroot bottleneck is negligibly small. This failure probability will be covered by a portion of the PAC confidence parameter $\delta$.

It remains to show that LearnTree will with high probability choose a bottleneck at each stage of the recursion in the first phase, and that Hancock's algorithm can be used to efficiently learn $\mathcal{T}_{d,n}$-random trees of depth $\frac{1}{2}\log n$ with high probability. We address the second point first in the next section.

## 6. Learning random ($\frac{1}{2}\log n$)-depth trees.

We stop the recursion in LearnTree at depth $\frac{1}{2}\log n$ because our analysis depends on trees being somewhat deep. So we use another method for learning random trees of depth less than $\frac{1}{2}\log n$, which is based on the following lemma plus the UnikDTLearn algorithm due to Hancock [7].

Recall that a decision tree $T$ is *read-$k$* if each variable labels at most $k$ nodes in $T$. We again write $\mathcal{T}_{d,n}$ to represent any of our three random tree models. We prove the following easy lemma in Appendix B.

LEMMA 6.1. *Let $r = ((1-\epsilon)\log n) - 2$ for some constant $\epsilon > 0$. Let $C$ be any constant. Then we have $\Pr_{T \in \mathcal{T}_{r,n}}[T \text{ is not read-}k] \leq 1/n^C$ for $k = (C+2)/\epsilon$.*

Thus, if $r = (\frac{1}{2}\log n)$, then for any constant $C$ we may take $k = 8C + 16$, and with probability at least $1 - \frac{4}{(n-2)^C}$ a tree $T$ drawn from $\mathcal{T}_{d,n}$ is read-$k$ (since each of the four subtrees of depth $r-2$ is not read-$(2C+4)$ with probability at most $1/(n-2)^C$).

Hancock [7] has given an algorithm UnikDTLearn and shown that it (or more precisely, a version which takes $k$ as an input along with the parameters given earlier) efficiently learns read-$k$ trees with respect to the uniform distribution, producing a decision tree (not necessarily read-$k$) as its hypothesis. Given any constant $k$, his algorithm terminates in time polynomial in $n$, $1/\epsilon$, and $1/\delta$, regardless of whether or not the target function $f$ is actually a read-$k$ decision tree. So our version of UnikDTLearn($n, \epsilon, \delta, EX(T, \mathcal{U})$) will begin by finding the smallest integer $C$ such that $1/n^C \leq \delta/2$. If the $\delta$ originally provided to LearnTree is inverse polynomial in $n$, then this value $C$ will be a constant independent of $n$. Taking $k = 8C + 16$, this means that the target function provided to UnikDTLearn is a read-$k$ decision tree with probability at least $1 - \delta/2$. Then running Hancock's original UnikDTLearn with this value of $k$ and with $\delta/2$ as the confidence parameter will succeed at learning an $\epsilon$-approximating tree with probability at least $1-\delta/2$, for an overall success probability at the bottom of the recursion of $1 - \delta$. In short, we have the following lemma.

LEMMA 6.2. *If the function $f$ in the oracle $EX(f,\mathcal{U})$ in the call to UnikDTLearn in LearnTree is distributed according to $\mathcal{T}_{d,n}$, then UnikDTLearn returns a decision tree that $\epsilon$-approximates $f$ with probability (over the random choice of $T$ and the randomness in $EX(f,\mathcal{U})$) at least $1-\delta$.*

It remains to show that the first stage of the algorithm successfully finds a bottleneck variable with high probability given a decision tree drawn at random according to one of our tree models and with depth at least $\frac{1}{2}\log n$. Throughout the rest of the paper we thus have $d = \Theta(\log n), d \geq \frac{1}{2}\log n$. We will consider each model separately, beginning with the complete model.

**7. Identifying bottlenecks in the complete model $\mathcal{T}^C_{d,n}$.** Since we have already shown that any bottleneck makes an equally good root in the hypothesis, and since it is easily seen that all bottlenecks (including the root of $T$) will pass the test at line 8 of LearnTree, it remains to show the following: for each $i = 1,\dots,n$, if $x_i$ is *not* a bottleneck in a random tree $T$, then the probability that $x_i$ passes the test in line 8 is negligibly small.

Our general plan of attack is as follows: we will prove that if $x_1$ is *not* a bottleneck, then with $1 - \frac{1}{n^{\omega(1)}}$ probability there are many root-to-leaf paths in $T$ that do not include $x_1$. We then argue that, conditioned on there being many such paths, among these preleaves there is a collection $C$ satisfying the condition of Lemma 3.3 which has $|C| = \omega(\log n)$. Combining this with the Fourier properties of random decision trees derived earlier gives us our result.

More precisely, the argument is as follows. Let $S$ be a random variable which denotes the number of $x_1$-free paths from the root to a preleaf in $T \in \mathcal{T}^C_{d,n}$. (Note that each such path ends at a depth-$d$ preleaf since we are in the complete model. Note also that since we are in the complete model, $S > 0$ if and only if $x_1$ is not a bottleneck.) We will prove the following lemmas in Appendix C.

LEMMA 7.1. *For $0 \leq d \leq n-1$ we have* $\Pr_{T\in\mathcal{T}^C_{d,n}}[S=0] \leq \frac{1}{n-d}$.

LEMMA 7.2. *For any value $1 \leq k \leq (\log n)^{3/2}$ we have* $\Pr[S=k] = 1/n^{\omega(1)}$.

LEMMA 7.3. *Let $T$ be drawn from $\mathcal{T}^C_{d,n}$ conditioned on its having some set of $(\log n)^{3/2}$ preleaves at depth $d$, each of which has no $x_1$-labeled node as an ancestor. Then with probability $1 - 1/n^{\omega(1)}$ there is a set $C$ of $(\log n)^{5/4}$ preleaves at depth $d$, each labeled with a distinct variable, each of which has no ancestor labeled with $x_1$ or with a variable that labels any element of $C$.*

From these lemmas it is easy to prove that each nonbottleneck will pass the test at line 8 with negligible probability.

THEOREM 7.4. *Let $T \in \mathcal{T}^C_{d,n}$, where $d = \Theta(\log n), d \geq \frac{1}{2}\log n$. If $x_1$ is not a bottleneck, then the probability that $x_1$ passes the test in line 8 is $1/n^{\omega(1)}$.*

*Proof.* Since $S = 0$ if and only if $x_1$ is a bottleneck, we have

$$\begin{aligned}\Pr_{T\in\mathcal{T}^C_{d,n}}[S < (\log n)^{3/2} \mid x_1 \text{ is not a bottleneck}] &= \frac{\Pr[S < (\log n)^{3/2} \ \&\ S > 0]}{\Pr[S>0]} \\ &= \frac{\Pr[1 \leq S < (\log n)^{3/2}]}{\Pr[S>0]} \\ &= 1/n^{\omega(1)},\end{aligned}$$

where the last equality follows from Lemmas 7.1 and 7.2. Thus we may assume that $S \geq (\log n)^{3/2}$. Lemma 7.3 now implies that there is a set $C$ of $(\log n)^{5/4}$ preleaves with the stated properties. Now we observe that if a preleaf belongs to $C$, then under any restriction $x_1 \leftarrow b$, the preleaf will still occur at depth $d$ with the desired property (that no variable labeling any node of $C$ occurs as an ancestor of any node of $C$) in the tree resulting from the restriction. Thus by Lemma 3.3, the probability that all variables labeling nodes in $C$ have even coefficients in the restricted tree is at most $1/2^{(\log n)^{5/4}} = 1/n^{\omega(1)}$. Hence $x_1$ passes the test at line 8 with negligible probability, and the theorem is proved. ∎

Combined with our earlier remarks, this establishes the following theorem.

THEOREM 7.5. *For any $n$, any polynomial $p(\cdot)$, any $\delta > 1/p(n)$, and any $\epsilon > 0$, algorithm LearnTree will with probability at least $1-\delta$ (over a random choice of tree $T$ from $\mathcal{T}^C_{d,n}$ and the randomness of the example oracle) produce a hypothesis decision tree $T'$ that $\epsilon$-approximates the target with respect to the uniform distribution. LearnTree runs in time polynomial in $n$ and $1/\epsilon$.*

*Proof.* By Lemma 6.2, the base case of the algorithm will succeed with probability at least $1-\delta$ as long as it is run on a tree drawn from $\mathcal{T}^C_{c,n}$ for some $c \leq \frac{1}{2}\log n$. In the recursive phase, all first-order Fourier coefficients will be computed exactly with probability at least $1-\delta/4$. Furthermore, assuming that the coefficients are correctly computed, every bottleneck variable will pass the test at line 8 of LearnTree, and by the preceding theorem the probability is negligible that any nonbottleneck variable will pass this test. Thus, in the recursive phase of the algorithm, with probability at least $1-\delta/4$ a bottleneck variable will be chosen by the test. By the arguments of section 5, the two functions obtained by restricting on either value of this bottleneck variable will both be distributed according to $\mathcal{T}^C_{d,n}$. Therefore, the two recursive calls to LearnTree will succeed with probability at least $1-\delta/2$, so that overall the recursive phase succeeds with probability at least $1-\delta$. Furthermore, it is easy to see that the tree returned by the recursive phase will be an $\epsilon$-approximator to the target if each of the subtrees returned by the recursive call is an $\epsilon$-approximator. Finally, for $d = O(\log n)$, the number of recursive calls is clearly polynomially bounded, and thus the algorithm runs in the time claimed given the previously mentioned bounds on UnikDTLearn and FCExact. ∎

**8. Identifying bottlenecks in the balanced model $\mathcal{T}^B_{d,n}$.** We first need an analogue of Lemma 3.3 for the balanced model. Let $T$ be a decision tree drawn from $\mathcal{T}^B_{d,n}$. Let $T'$ be the tree that results from applying the restriction $x_1 \leftarrow 1$ to $T$; i.e., $T'$ is obtained from $T$ by replacing each occurrence of $x_1$ in $T$ with its right subtree.

As in Lemma 3.3, each internal node of $T'$ has a "parity" which is defined exactly as in the proof of Lemma 3.3, and each variable $x_i$ has a parity which equals the parity of the parity of all nodes labeled $x_i$ in $T'$. It is easily seen that rule 1 contributes to the parity of $x_i$ in $T'$ precisely once for each occurrence of $x_i$ as a preleaf in $T$ which lies on an $x_1$-free path from the root in $T$; let $N_{i,1}$ denote the number of such occurrences. Rule 2 contributes to the parity of $x_i$ in $T'$ precisely once for each occurrence of $x_1$ as a preleaf in $T$ which lies on a path from the root in $T$ which contains $x_i$; let $N_{i,2}$ denote the number of such occurrences. (Rule 3 can never be satisfied since each pair of sibling leaf bits in $T'$ must have opposite signs.) Note that the values of $N_{i,1}$ and $N_{i,2}$ are independent of whether we defined $T'$ by the restriction $x_1 \leftarrow -1$ or $x_1 \leftarrow 1$ in the previous paragraph.

As in section 7, to prove that LearnTree succeeds for random $T$ drawn from $\mathcal{T}^B_{d,n}$, we must show that if $x_1$ is not a bottleneck in a random $T$ drawn from $\mathcal{T}^B_{d,n}$, then the probability that $x_1$ passes the test in line 8 is $1/n^{\omega(1)}$. From the discussion of the previous paragraph, this is equivalent to showing that

$$\Pr[N_{i,1} + N_{i,2} \text{ is even for all } i = 2, \dots, n] = 1/n^{\omega(1)}. \tag{8.1}$$

We prove this as follows. As in section 7 let $S$ be a random variable which denotes the number of $x_1$-free paths from the root to a preleaf in $T$. (The difference is that now $T$ is drawn from $\mathcal{T}^B_{d,n}$ instead of $\mathcal{T}^C_{d,n}$; however, these two distributions are identical as regards the internal nodes of a tree drawn from one of them.) By Lemmas 7.1 and 7.2 we know that conditioned on $x_1$ not being a bottleneck in $T$, we have $\Pr_{T\in\mathcal{T}^B_{d,n}}[S > (\log n)^{3/2}] = 1 - 1/n^{\omega(1)}$ (these lemmas are easily seen to hold for $\mathcal{T}^B_{d,n}$ as well as $\mathcal{T}^C_{d,n}$). We thus may safely assume that there are $S \geq (\log n)^{3/2}$ many root-to-preleaf $x_1$-free paths in $T$.

Now fix any set of $S$ preleaf nodes in a complete binary tree of depth $d$. Fix a labeling of variables for each node on each of these paths except for the preleaves themselves (where the labeling is nonredundant and never uses $x_1$); call this labeling $L$. Fix any bit string $v = v_2, \dots, v_n \in \{0,1\}^{n-1}$ corresponding to the parities of $N_{2,2}, \dots, N_{n,2}$. We will show that conditioned on $L$ being the labeling of the set of nonpreleaf nodes on the $x_1$-free paths in $T$, the probability (over the random labeling of the $S$ preleaf nodes) that each variable $x_2, \dots, x_n$ occurs with the "right" parity $v_2, \dots, v_n$ among the $S$ preleaf nodes is $1/n^{\omega(1)}$. (In other words, we will show that regardless of what parities $N_{2,2}, \dots, N_{n,2}$ have, the probability that each $N_{i,1}$ exactly matches the corresponding parity of $N_{i,2}$ is $1/n^{\omega(1)}$.) This suffices to establish (8.1). Thus, it suffices to prove the following lemma (the proof is given in Appendix D).

LEMMA 8.1. *Fix $S \geq 2(\log n)^{3/2}$. For $i = 1, \dots, S$ fix $P_i$ to be some set of exactly $n - d$ "permissible" elements of $\{1, \dots, n\}$. Fix $v = v_1, \dots, v_n \in \{0,1\}^n$. Let $P$ denote $P_1 \times P_2 \times \cdots \times P_S$, so $|P| = (n-d)^S$. Given $z = (z_1, \dots, z_S) \in P$, for $j = 1, \dots, n$ let $par_j(z)$ denote the number of occurrences modulo 2 of $j$ in $z$. Then we have $\Pr_{z\in P}[par_j(z) = v_j$ for all $j = 1, \dots, n] = \frac{1}{n^{\omega(1)}}$.*

We thus have the following balanced model analogues of Theorems 7.4 and 7.5.

THEOREM 8.2. *Let $T \in \mathcal{T}^B_{d,n}$, where $d = \Theta(\log n), d \geq \frac{1}{2}\log n$. If $x_1$ is not a bottleneck, then the probability that $x_1$ passes the test in line 8 is $1/n^{\omega(1)}$.*

THEOREM 8.3. *For any $n$, any polynomial $p(\cdot)$, any $\delta > 1/p(n)$, and any $\epsilon > 0$, algorithm LearnTree will with probability at least $1-\delta$ (over a random choice of tree $T$ from $\mathcal{T}^B_{d,n}$ and the randomness of the example oracle) produce a hypothesis decision tree $T'$ that $\epsilon$-approximates the target with respect to the uniform distribution. LearnTree runs in time polynomial in $n$ and $1/\epsilon$.*

## 9. Identifying bottlenecks in the uniform model $\mathcal{T}^U_{d,n}$.

As in section 7, it suffices to prove that if $x_1$ is not a bottleneck in a random tree $T$ drawn from $\mathcal{T}^U_{d,n}$, then $\Pr[x_1$ passes the test in line 8$] = 1/n^{\omega(1)}$. Our basic approach is similar to section 7, but there are some differences, as shown below.

We now let $S$ be a random variable which denotes the number of $x_1$-free paths from the root to a preleaf of depth $d$ in a random $T$ drawn from $\mathcal{T}^U_{d,n}$. Note that in $\mathcal{T}^U_{d,n}$ it *is* possible to have $S = 0$ even if $x_1$ is not a bottleneck; this can happen if there exist $x_1$-free root-to-preleaf paths in $T$ but all such pre-leaves have depth less than $d$. We will establish the following lemma in Appendix E.

LEMMA 9.1. *For any value $0 \leq k \leq (\log n)^{3/2}$ we have $\Pr_{T\in\mathcal{T}^U_{d,n}}[S = k \mid x_1$ is not a bottleneck $] = 1/n^{\omega(1)}$.*

With this lemma we can prove a uniform model analogue of Theorem 7.4.

THEOREM 9.2. *Let $T \in \mathcal{T}^U_{d,n}$, where $d = \Theta(\log n), d \geq \frac{1}{2}\log n$. If $x_1$ is not a bottleneck, then the probability that $x_1$ passes the test in line 8 is $1/n^{\omega(1)}$.*

*Proof.* Since $x_1$ is not a bottleneck, we have $S \geq (\log n)^{3/2}$ with probability $1 - \frac{1}{n^{\omega(1)}}$ by Lemma 9.1. We now observe that Lemma 7.3 holds identically (with the same proof) if in its statement $T$ is drawn from $\mathcal{T}^U_{d,n}$ and we further condition on $x_1$ not being a bottleneck in $T$. We thus have that with $1 - \frac{1}{n^{\omega(1)}}$ probability there is a set $C$ of $(\log n)^{5/4}$ variables with the properties stated in Lemma 7.3. The rest of the proof follows exactly the proof of Theorem 7.4. □

Now a nearly[3] identical proof to that of Theorem 7.5 gives us our main learning result for uniform random trees.

THEOREM 9.3. *For any $n$, any polynomial $p(\cdot)$, any $\delta > 1/p(n)$, and any $\epsilon > 0$, algorithm LearnTree will with probability at least $1-\delta$ (over a random choice of tree $T$ from $\mathcal{T}^U_{d,n}$ and the randomness of the example oracle) produce a hypothesis decision tree $T'$ that $\epsilon$-approximates the target with respect to the uniform distribution. LearnTree runs in time polynomial in $n$ and $1/\epsilon$.*

## 10. Conclusions and future work.

We have given positive results for learning several natural models of random log-depth decision trees under the uniform distribution. Many interesting questions remain about related models of average-case learning:

- Can similar results be established for natural models of random decision trees of polynomial size (as opposed to logarithmic depth)?
- Can similar results be established for random disjunctive normal form (DNF) formulas or random monotone DNF?
- Can our results be extended to learning under a broader class of distributions?
- Can similar ideas be used to learn with high probability when the target is drawn randomly from an interesting nonuniform distribution over log-depth trees?

It seems possible that progress in these directions could eventually lead to useful practical algorithms.

## Appendix A. Proof of Lemma 5.3.

The proof is by induction on the depth $d$ of $T$. If $d < 1$, then the only possible bottleneck in $T$ is the root and the lemma holds vacuously. For the inductive step, consider a tree $T$ of depth $d \geq 1$.

- If $x_i$ is the root of $T$, then $T'$ is just $T$.
- If $x_i$ is not the root of $T$, then $T$ has some variable $x_j$ at the root with left subtree denoted by $T_1$ and right subtree $T_2$. Since $x_i$ is a bottleneck in $T$ but is not the root of $T$, $x_i$ must be a bottleneck in $T_1$ and in $T_2$. By the induction hypothesis there are trees $T_1', T_2'$ each of which has $x_i$ at its root and such that each is structurally equivalent to $T_1, T_2$, respectively. By definition of structural equivalence, the tree $T''$ with $x_j$ at its root and $T_1'$, $T_2'$ as the left and right child subtrees of the root is structurally equivalent to $T$. Performing a root swap on $T''$ gives the required $T'$.

## Appendix B. Proof of Lemma 6.1.

We prove the lemma for the complete tree model $\mathcal{T}^C_{d,n}$; the proof holds unchanged for $\mathcal{T}^B_{d,n}$ and is easily adapted to $\mathcal{T}^U_{d,n}$. We

[3]We need to fold the distribution irregularity noted in section 5 into $\delta$ at the obvious point in the proof.

have

$$\Pr[T \text{ is not read-}k] \leq n \cdot \Pr[x_1 \text{ occurs } k' > k \text{ times in } T \text{ for some } k'].$$

Since $T$ is of depth $r$, there are $2^{r+1}-1$ occurrences of variables in $T$. The probability that $x_1$ occurs $k'$ times in $T$ is at most $\binom{2^{r+1}-1}{k'} \cdot \left(\frac{1}{n-r}\right)^{k'}$. This is because there are at most $\binom{2^{r+1}-1}{k'}$ possible sets of locations for the $k'$ occurrences of $x_1$ in $T$, and for each location the probability that $x_1$ actually occurs there is at most $1/(n-r)$ (even conditioned on any labeling of the other nodes of the tree), since each location has at most $r$ ancestors.

Since $\binom{a}{b} \leq a^b$ and $1/(n-r) < 2/n$, we can upper bound this probability by

$$2^{(r+1)k'} \cdot 2^{k'}/n^{k'} = 2^{(r+2)k'}/n^{k'} = n^{(1-\epsilon)k'}/n^{k'} = 1/n^{\epsilon k'} \leq 1/n^{\epsilon k}.$$

Since there are at most $n$ possible values of $k' \geq k$ (no variable can occur more than $n$ times since the size of the tree is less than $n$) all in all we have

$$\Pr_{T \in \mathcal{T}_{r,n}}[T \text{ is not read-}k] \leq 1/n^{\epsilon k - 2} = 1/n^C. \qquad \square$$

**Appendix C. Proofs of Lemmas 7.1, 7.2, and 7.3.** We first prove Lemma 7.1. Let us write $p_{d,n}$ to denote the probability that $S = 0$ for a random $T$ drawn from $\mathcal{T}^C_{d,n}$, i.e.,

$$p_{d,n} = \Pr_{T \in \mathcal{T}^C_{d,n}}[x_1 \text{ is a bottleneck in } T].$$

Lemma 7.1 follows directly from the following.

PROPOSITION C.1. *For* $0 \leq d \leq n-1$ *we have* $p_{d,n} \leq \frac{1}{n-d}$.

*Proof.* Clearly $p_{0,n} = \frac{1}{n}$. For $d \geq 1, n \geq 1$ we have $p_{d,n} = \frac{1}{n} + \frac{n-1}{n}(p_{d-1,n-1})^2$. This is because with probability $\frac{1}{n}$ the root is $x_1$. With probability $\frac{n-1}{n}$ the root is some $x_j \neq x_1$, in which case each of the subtrees of the root is drawn from $\mathcal{T}^C_{d-1,n-1}$, and $x_1$ is a bottleneck if and only if it is a bottleneck in each of these two subtrees.

Fix any $m > 0$. We prove that for all $d \geq 0$ we have $p_{d,d+m} \leq \frac{1}{m}$; the proof is by induction on $d$. The base case holds since $p_{0,m} = \frac{1}{m}$. For the induction step, we have

$$\begin{aligned} p_{d+1,d+m+1} = \frac{1}{d+m+1} + \frac{d+m}{d+m+1}(p_{d,d+m})^2 &\leq \frac{1}{d+m+1} + \frac{d+m}{d+m+1}\left(\frac{1}{m}\right)^2 \\ &= \frac{m^2+m+d}{m^2(m+d+1)}. \end{aligned}$$

This is at most $\frac{1}{m}$ if and only if $m^3 + dm + m^2 \leq m^3 + dm^2 + m^2$, which is true since $m \geq 1$, so the proposition is proved. $\square$

We will prove Lemma 7.2 in a moment using Lemma C.2 below. First, a definition and some introductory analysis.

Let $p_{k,d,n}$ denote $\Pr_{T \in \mathcal{T}^C_{d,n}}[S = k]$. For $k \geq 1$ and $d \geq 1$ we have

$$p_{k,d,n} = \frac{n-1}{n} \sum_{i=0}^{k} p_{i,d-1,n-1} p_{k-i,d-1,n-1}. \tag{C.1}$$

To see this, note that there are exactly $k$ $x_1$-free root-to-preleaf paths in $T$ if and only if (1) the root is some variable other than $x_1$, and (2) the left and right subtrees (each of which is drawn from $\mathcal{T}^C_{d-1,n-1}$) have $i$ and $k-i$ $x_1$-free root-to-preleaf paths, respectively, for some $1 \le i \le k$. For the base cases, we have $p_{c,0,n} = 0$ for $c \ge 2$ since it is impossible to have two paths to preleaves in a tree of depth 0. $p_{1,0,n} = \frac{n-1}{n}$ since there is exactly one $x_1$-free path as long as the root is not $x_1$. $p_{0,0,n} = \frac{1}{n}$ by the same reasoning. Finally, $p_{0,d,n} \le \frac{1}{n-d}$ by Lemma 7.1.

The following lemma is proved in section C.1.

LEMMA C.2. *Let $c = \Theta(\log n), c \ge \frac{3}{8}\log n$, and $\ell \le poly(n)$. Then*

$$p_{\ell,c,n} \le t(n) + \sum_{j=1}^{(\log n)^{1/3}} \sum_{i=(1/4)\log n+1}^{\ell-(1/4)\log n-1} p_{i,c-j,n-j}, \tag{C.2}$$

*where $t(n) = 1/n^{\omega(1)}$.*

*Proof of Lemma* 7.2. Recall that $k \le (\log n)^{3/2}$, $d = \Theta(\log n)$, and $d \ge \frac{1}{2}\log n$. By Lemma C.2 we have

$$p_{k,d,n} \le t(n) + \sum_{j=1}^{(\log n)^{1/3}} \sum_{i=(1/4)\log n+1}^{k-(1/4)\log n-1} p_{i,d-j,n-j}. \tag{C.3}$$

We now repeatedly apply Lemma C.2 to the right-hand side of inequality (C.3). The key observation is that each time we apply Lemma C.2 to bound some $p_{\ell,c,n'}$ by the right side of (C.2), the first subscript ($\ell$) decreases by at least $\frac{1}{4}\log n$ in every new occurrence of $p_{\cdot,\cdot,\cdot}$. Hence the "depth" of this repeated replacement will be at most $4(\log n)^{1/2}$ (since $k \le (\log n)^{3/2}$), at which point the summation over $i$ in the right-hand side of (C.2) will be empty.

We now observe that each application of Lemma C.2 replaces one $p_{\cdot,\cdot,\cdot}$ with at most $(\log n)^{1/3} \cdot (\log n)^{3/2} < (\log n)^2$ new $p_{\cdot,\cdot,\cdot}$'s. Since the replacement depth is at most $4(\log n)^{1/2}$ and $t(n) = 1/n^{\omega(1)}$, it follows that

$$p_{k,d,n} \le \frac{1}{n^{\omega(1)}} \cdot \left((\log n)^2\right)^{4(\log n)^{1/2}} = \frac{1}{n^{\omega(1)}} \cdot 2^{8(\log n)^{1/2}\log\log n} = \frac{1}{n^{\omega(1)}},$$

and Lemma 7.2 is proved. $\square$

Now we prove Lemma 7.3.

*Proof of Lemma* 7.3. Fix any set $R$ of $(\log n)^{3/2}$ preleaf nodes in a complete binary tree structure of depth $d$. Fix any nonredundant labeling of all of the ancestors of all of these preleaves which does not use $x_1$ anywhere. Now each labeling of the nodes in $R$ which does not use $x_1$ and maintains nonredundancy is equally as likely under the conditioning of the lemma. Note that for each node in $R$ there are $n-d-1$ legal labelings (since the label must not use $x_1$ or any of the $d$ ancestors of the node).

Consider a random legal labeling of the nodes in $R$. Partition the nodes of $R$ into $(\log n)^{5/4}$ disjoint subsets $R_1, \dots, R_{(\log n)^{5/4}}$ each of size $(\log n)^{1/4}$. Let $F$ denote a set of "forbidden" labels; initially $F$ is the set of all variables which label ancestors of nodes in $R$ (plus $x_1$). Let $F_0$ denote the size of this initial set, so initially we have $|F| = F_0 \le 1 + d(\log n)^{3/2} = O((\log n)^{5/2})$. We consider the subsets $R_1, \dots$ in turn. The probability that every node in $R_1$ is assigned a forbidden label is at most $\left(\frac{F_0}{n-d-1}\right)^{(\log n)^{1/4}} = 1/n^{\omega(1)}$. Thus we may suppose that there is some preleaf $v_1 \in R_1$ which receives a nonforbidden label; we add this label to $F$. Now the probability that every node in $R_2$ receives a forbidden label is at most $\left(\frac{F_0+1}{n-d-1}\right)^{(\log n)^{1/4}} = 1/n^{\omega(1)}$, so we may suppose that there is some preleaf $v_2 \in R_2$ which receives a nonforbidden label; we add this label to $F$. Continuing in this fashion for $(\log n)^{5/4}$ steps, and noting that $|F|$ never exceeds $O((\log n)^{5/2})$, we have that with probability $1 - 1/n^{\omega(1)}$ there is a set $v_1, \ldots, v_{(\log n)^{5/4}}$ of nodes each of which receives a nonforbidden label. This set is easily seen to satisfy the desired conditions for $C$. ∎

**C.1. Proof of Lemma C.2.** Our proof of Lemma C.2 will use the following intermediate lemma. Note that we allow a slightly weaker bound on $d$ than usual in this lemma; we will need this slightly weaker bound later.

LEMMA C.3. *For any value* $1 \le k \le \frac{1}{4}\log n$ *and any value* $d \ge \frac{1}{3}\log n$ *we have* $p_{k,d,n} = \Pr_{T \in \mathcal{T}^C_{d,n}}[S = k] = 1/n^{\omega(1)}$.

*Proof.* We first consider the case $k = 1$. There are exactly $2^d$ possible locations (preleaves) where an $x_1$-free path from the root to a preleaf could end. Consider any such location. In order for this to be the only $x_1$-free path to a preleaf in $T$, it must be the case that every node on this path (except the root) has the property that the subtree rooted at its sibling has $x_1$ as a bottleneck. These $d$ subtrees are clearly disjoint; the one at depth $\ell$ is drawn from $\mathcal{T}^C_{d-\ell,n-\ell}$ (over a suitable set of $n - \ell$ variables which includes $x_1$ since the path is $x_1$-free), and hence by Lemma 7.1 each subtree has $x_1$ as a bottleneck with probability at most $\frac{2}{n}$. Thus $\Pr[S = 1]$ is at most $2^d \cdot \left(\frac{2}{n}\right)^d = \left(\frac{4}{n}\right)^d$, which is $1/n^{\omega(1)}$ since $d \ge \frac{1}{3}\log n$.

The general case for any $1 \le k \le \frac{1}{4}\log n$ is similar. We use the following fact, which we prove later.

FACT 1. *Fix any set of $k$ root-to-preleaf paths in $T$. Let $N$ be the number of subtrees of $T$ which are rooted at an internal node and* (1) *are not rooted on any of these $k$ paths, but* (2) *have their parent on one of these $k$ paths. Then $N \ge d - \log k$.*

There are $\binom{2^d}{k}$ possible sets of $k$ preleaves where the $x_1$-free paths might end. As in the case $k = 1$, each subtree as in Fact 1 must have $x_1$ as a bottleneck, but as in the $k = 1$ case each such subtree has $x_1$ as a bottleneck with probability at most $2/n$. Thus the probability that $S = k$ is at most (by Fact 1)

$$\binom{2^d}{k} \cdot \left(\frac{2}{n}\right)^{d-\log k} \le 2^{dk}\left(\frac{2}{n}\right)^{d-\log k} \le n^{d/4} \cdot \left(\frac{2}{n}\right)^d \cdot n^{\log k} = n^{\log k}\left(\frac{2}{n^{3/4}}\right)^d,$$

where the second inequality uses $k \le \frac{1}{4}\log n$. This is $1/n^{\omega(1)}$ since $d \ge \frac{1}{3}\log n$ and $k \le \frac{1}{4}\log n$. ∎

*Proof of Fact* 1. It is clear that there are exactly $2^d - k$ preleaves contained in the desired subtrees of $T$. Each subtree contains $2^i$ of these preleaves for some $i$, and clearly different subtrees have disjoint sets of preleaves. Since the binary representation of $2^d - k$ starts with $d - \log k$ ones, there must be at least $d - \log k$ such subtrees (it is impossible to add up $t$ powers of 2 and get a binary number with more than $t$ ones). ∎

Now we prove Lemma C.2. From the recursive equation (C.1) we have

$$p_{\ell,c,n} \le 2p_{0,c-1,n-1}p_{\ell,c-1,n-1} + \sum_{i=1}^{\ell-1} p_{i,c-1,n-1}p_{\ell-i,c-1,n-1}$$

$$\le \frac{4}{n}p_{\ell,c-1,n-1} + \sum_{i=1}^{\ell-1} p_{i,c-1,n-1}p_{\ell-i,c-1,n-1}, \tag{C.4}$$

where the last inequality holds (with room to spare) by Lemma 7.1 since $c = \Theta(\log n) <$

$n/2$. Repeatedly applying (C.4), we have

$$\begin{aligned}p_{\ell,c,n} &\le \left(\frac{4}{n}\right)^2 p_{\ell,c-2,n-2} + \frac{4}{n}\sum_{i=1}^{\ell-1} p_{i,c-2,n-2}p_{\ell-i,c-2,n-2} + \sum_{i=1}^{\ell-1} p_{i,c-1,n-1}p_{\ell-i,c-1,n-1}\\ &\le \cdots\\ &\le \left(\frac{4}{n}\right)^c p_{\ell,0,n-c} + \sum_{j=1}^{c}\left(\frac{4}{n}\right)^{j-1}\sum_{i=1}^{\ell-1} p_{i,c-j,n-j}p_{\ell-i,c-j,n-j}.\end{aligned}$$

Since each value $p_{\cdot,\cdot,\cdot}$ is a probability, it is easy to see that for any value of $j$ the inner sum over $i$ is at most $\ell =$poly$(n)$. Recalling that $c \ge \frac{3}{8}\log n$, we may truncate the sum over $j$ at (say) $(\log n)^{1/3}$ and thus have

$$\begin{aligned}p_{\ell,c,n} &\le \frac{1}{n^{\omega(1)}} + \sum_{j=1}^{(\log n)^{1/3}}\sum_{i=1}^{\ell-1} p_{i,c-j,n-j}p_{\ell-i,c-j,n-j}\\ &\le \frac{1}{n^{\omega(1)}} + \sum_{j=1}^{(\log n)^{1/3}}\left[2\sum_{i=1}^{(1/4)\log n} p_{i,c-j,n-j} + \sum_{i=(1/4)\log n+1}^{\ell-(1/4)\log n-1} p_{i,c-j,n-j}\right].\end{aligned}$$

Since $c-(\log n)^{1/3}$ is at least $(1/3)\log n$, Lemma C.3 implies that the first sum over $i$ inside the brackets is $1/n^{\omega(1)}$ for all $j = 1,\ldots,(\log n)^{1/3}$. We thus have

$$p_{\ell,c,n} \le \frac{1}{n^{\omega(1)}} + \sum_{j=1}^{(\log n)^{1/3}}\sum_{i=(1/4)\log n+1}^{\ell-(1/4)\log n-1} p_{i,c-j,n-j}$$

as desired, and Lemma C.2 is proved.

**Appendix D. Proof of Lemma 8.1.** We say that $z \in P$ *yields* $v \in \{0,1\}^n$ if $par_j(z) = v_j$ for all $j$. Let $V \subseteq P$ denote the set of all $z$ which yield $v$; thus our goal is to show that $|V|/|P| = 1/n^{\omega(1)}$. We do this by defining a mapping $M$ with the following properties:

1. $M$ assigns to each $z \in V$ a set of $(n-d-2\log n)^{\log n}$ strings all of which are in $P - V$.
2. For each $z \in V$, for any $x \in M(z)$, the number of $z' \in V$ such that $x \in M(z')$ is at most $(1/n^{\omega(1)}) \cdot (n-d-2\log n)^{\log n}$.

These properties imply that $|V|/|P-V| = 1/n^{\omega(1)}$, which establishes the lemma.

The mapping $M$ is defined as follows: given $z \in V$, the elements of $M(z)$ are those $x \in P$ which satisfy the following conditions:

- $x_i = z_i$ for all $i = 1,\ldots,S-\log n$.
- For $i = S-\log n+1,\ldots,S$, coordinate $x_i$ is an element of $P_i$ which (a) does not occur in the last $\log n$ coordinates of $z$, and (b) does not equal $x_j$ for any other value $j \in S-\log n+1,\ldots,S$.

There are at least $(n-d-2\log n)^{\log n}$ elements of $M(z)$ since the second condition above rules out at most $2\log n$ of the $n-d$ elements of $P_i$ for each $i$.

Fix some $z \in V$ and some $x \in M(z)$. We now show that there are few $z' \in V$ such that $x \in M(z')$. Note first that in order for $z'$ to have $x \in M(z')$ it must be the case that $z'_i = z_i$ for all $i = 1,\ldots,S-\log n$. Let $z^*$ denote this $S-\log n$ character prefix $z_1\ldots z_{S-\log n}$. We now show that only a small number of the $(n-d)^{\log n}$ possible completions $\sigma \in P_{S-\log n+1}\times\cdots\times P_S$ will be such that $z' = z^*\sigma$ belongs to $V$.

We prove this by showing that in fact only a small number of the $n^{\log n}$ possible completions $\sigma \in \{1,\dots,n\}^{\log n}$ will be such that $z^*\sigma$ belongs to $V$. To see this, note that in order for $z^*\sigma$ to belong to $V$, each $j \in \{1,\dots,n\}$ must occur either an even or an odd number of times in $\sigma$ (depending on whether $par_j(z^*)$ does or does not match $v_j$). Now we observe that the probability that these $n$ parity conditions are all satisfied by a random $\sigma \in \{1,\dots,n\}^{\log n}$ is precisely the probability that a uniform random walk in the Boolean cube $\{0,1\}^n$ ends up, after precisely $\log n$ steps, at some particular vertex $w \in \{0,1\}^n$ (where the walk starts at $0^n$ and proceeds to a randomly chosen neighbor of the current node at each step). Since $d = \Theta(\log n)$ we have $(n-d-2\log n)^{\log n}/n^{\log n} = \Theta(1)$, and thus Lemma 8.1 follows from the following elementary fact, which for completeness we now prove.

PROPOSITION D.1. *For all $w \in \{0,1\}^n$, the probability that a uniform random walk of precisely $\log n$ steps starting at $0^n$ ends at $w$ is $1/n^{\omega(1)}$.*

*Proof.* Let $|w|$ denote the number of nonzero coordinates in $w$. Let $p_w$ denote the probability that the walk ends at $w$. We first observe that if $|w| = |w'|$, then by symmetry $p_w = p_{w'}$. Thus for any $w$ such that $|w| \geq \frac{1}{3}\log n$ we clearly have $p_w \leq 1/\binom{n}{(1/3)\log n} = 1/n^{\omega(1)}$. Thus we may suppose that $|w| \leq \frac{1}{3}\log n$. Any walk of $\log n$ steps which ends at such a $w$ must select at most $\frac{2}{3}\log n$ distinct indices from $\{1,\dots,n\}$ in total, since at most $\frac{1}{3}\log n$ of the indices selected are selected exactly once. The number of possible $\log n$ step walks which select at most $\frac{2}{3}\log n$ distinct indices is at most $\binom{n}{(2/3)\log n} \cdot (2/3\log n)^{\log n}$, since there are $\binom{n}{(2/3)\log n}$ ways to choose the selected indices and then $(2/3\log n)^{\log n}$ ways to choose the walk using these indices. This is less than $n^{2/3\log n} \cdot (n^{1/6})^{\log n} = n^{(5/6)\log n}$, and hence such walks occur with total probability at most $1/n^{(1/6)\log n}$ since there are $n^{\log n}$ possible walks in total. ∎

**Appendix E. Proofs of Lemmas 9.1, 5.5, and 5.7.** Before proving Lemma 9.1 we first establish some useful notation and observations.

Let $C_{d,n}$ denote the number of nonredundant decision trees over $\{x_1,\dots,x_n\}$ of depth at most $d$. (So $C_{-1,n} = 2$ since a single leaf can be either $-1$ or 1; $C_{0,n} = 2+4n$ since there are $4n$ possibilities for a depth-0 tree depending on the variable at the root and the two leaf bits; etc.) The following observation will be useful.

OBSERVATION 2. *Drawing a random $T$ from $\mathcal{T}^U_{d,n}$ is equivalent to generating $T$ via the following randomized process which we call $\texttt{MakeTree}_{d,n}$:*

- *With probability $1/C_{d,n}$ take $T$ to be the one-node tree $+1$ and halt. Likewise, with probability $1/C_{d,n}$ take $T$ to be the one-node tree $-1$ and halt.*
- *With probability $1 - 2/C_{d,n}$ pick a random variable from $x_1,\dots,x_n$ as the root of $T$. Construct its left and right subtrees by independently performing two calls to $\texttt{MakeTree}_{d-1,n-1}$, using for each call the set of $n-1$ variables which were not selected for the root.*

We write $q_{d,n}$ to denote $1 - \frac{2}{C_{d,n}}$; i.e., $q_{d,n}$ is the probability that a randomly drawn $T$ from $\mathcal{T}^U_{d,n}$ is nontrivial (recall that there are exactly two trivial trees).

We now write $p_{d,n}$ to denote the probability that a random $T$ drawn from $\mathcal{T}^U_{d,n}$ has $x_1$ as a bottleneck:

$$p_{d,n} \equiv \Pr_{T\in\mathcal{T}^U_{d,n}}[x_1 \text{ is a bottleneck}].$$

We have the following analogue of Lemma 7.1.

LEMMA E.1. *For all $0 \leq d \leq n-1$, $p_{d,n} \leq \frac{1}{n-d}$.*

*Proof.* The proof differs only slightly from that of Lemma 7.1. We now have $p_{0,n} = \frac{4}{4n+2} < \frac{1}{n}$ since exactly four of the $4n+2$ trees of depth at most 0 have $x_1$ as a bottleneck (i.e., as the root). For $d \geq 1, n \geq 1$ we now have

$$p_{d,n} = q_{d,n}\left(\frac{1}{n} + \frac{n-1}{n}(p_{d-1,n-1})^2\right). \tag{E.1}$$

To see this, note that $x_1$ is a bottleneck only if $T$ is nontrivial, which occurs with probability $q_{d,n}$. If $T$ is nontrivial, then with probability $\frac{1}{n}$ the root is $x_1$, in which case $x_1$ is a bottleneck. Otherwise, $x_1$ is a bottleneck if and only if $x_1$ is a bottleneck in the left and right subtrees of $T$, each of which is drawn from $\mathcal{T}^U_{d-1,n-1}$.

Comparing the initial conditions and recurrence relation for $p_{d,n}$ with those of Lemma 7.1 it is clear that the current $p_{d,n}$ is dominated by the earlier recurrence, and the lemma is proved. ◻

Now we can prove Lemma 9.1. We have that

$$\begin{aligned}\Pr_{T\in\mathcal{T}^U_{d,n}}[S = k \mid x_1 \text{ is not a bottleneck}] &= \frac{\Pr[S = k \;\&\; x_1 \text{ is not a bottleneck}]}{\Pr[x_1 \text{ is not a bottleneck}]}\\ &< 2\Pr[S = k \;\&\; x_1 \text{ is not a bottleneck}]\\ &\leq 2\Pr[S = k],\end{aligned}$$

where the first inequality is by Lemma E.1. Thus it suffices to prove the following two lemmas.

Lemma E.2. *For all* $1 \leq k \leq (\log n)^{3/2}$ *we have* $\Pr_{T\in\mathcal{T}^U_{d,n}}[S = k] = 1/n^{\omega(1)}$.

Lemma E.3. *For* $d \geq \frac{1}{2}\log n$, $d = \Theta(\log n)$, $\Pr_{T\in\mathcal{T}^U_{d,n}}[S = 0 \mid x_1$ *is not a bottleneck*$] = 1/n^{\omega(1)}$.

(Note that bounding $\Pr[S = 0 \mid x_1$ is not a bottleneck$]$ by $2\Pr[S = 0]$ is a bad idea since $S = 0$ whenever $x_1$ is a bottleneck, and this occurs with probability roughly $1/n$; hence we use the above approach of handling $S = 0$ separately by bounding the conditional probability directly.)

*Proof of Lemma* E.2. We closely imitate the proof of Lemma 7.2. Let $p_{k,d,n}$ now denote $\Pr_{T\in\mathcal{T}^U_{d,n}}[S = k]$. For $k \geq 1$ and $d \geq 1$ we now have

$$p_{k,d,n} = q_{d,n} \cdot \frac{n-1}{n} \cdot \sum_{i=0}^{k} p_{i,d-1,n-1}p_{k-i,d-1,n-1}. \tag{E.2}$$

(The $q_{d,n} \cdot \frac{n-1}{n}$ is present because in order for $S$ to be nonzero it must be the case that $T$ is nontrivial and that the root is not $x_1$. If this is the case, then the left and right subtrees (which, under this conditioning, are drawn from $\mathcal{T}^U_{d-1,n-1}$) must have $i$ and $k-i$ $x_1$-free paths from their respective roots to preleaves at depth $d-1$ in those subtrees.) As before we have $p_{c,0,n} = 0$ for $c \geq 2$ since there cannot be two paths to preleaves in a depth-0 tree. We have $p_{1,0,n} = q_{0,n} \cdot \frac{n-1}{n}$ since there is one $x_1$-free path if and only if the tree is nontrivial and the root is not $x_1$. Moreover, we have $p_{0,0,n} = \frac{6}{4n+2}$ since the only trees of depth at most 0 which have $S = 0$ are the four trees with $x_1$ at the root and the two trivial trees.

Finally, if $k = 0$ and $d > 0$, then we have

$$p_{0,d,n} = (1 - q_{d,n}) + q_{d,n} \cdot \frac{1}{n} + q_{d,n} \cdot \frac{n-1}{n}(p_{0,d-1,n-1})^2. \tag{E.3}$$

(We have that $S = 0$ if $T$ is trivial, or if $T$ is nontrivial and $x_1$ is the root, or if $T$ is nontrivial, some other variable is the root, and both subtrees have no $x_1$-free paths to preleaves at depth $d-1$.) Equation (E.3), combined with the fact that $1 - q_{d,n} < \frac{1}{n}$ for $d \geq 0$, implies that for $d \geq 1$ we have

$$p_{0,d,n} \leq \frac{2}{n} + \frac{n-1}{n}(p_{0,d-1,n-1})^2.$$

A straightforward analysis of this recurrence shows that $p_{0,d,n} < \frac{3}{n}$ for all $d = O(\log n)$.

Now let $\tilde{p}_{k,d,n}$ be defined by the same base case conditions (for $k = 0$ or $d = 0$) that we have just given, but be defined for $k \geq 1, d \geq 1$ by the rule from section C, i.e.,

$$\tilde{p}_{k,d,n} = \frac{n-1}{n}\sum_{i=0}^{k}\tilde{p}_{i,d-1,n-1}\tilde{p}_{k-i,d-1,n-1}.$$

(Note that these base conditions differ only slightly from the base conditions on $p_{k,d,n}$ in section C; the bound $\frac{3}{n}$ which we have on $\tilde{p}_{0,d,n}$ is slightly weaker than the $\frac{2}{n}$ bound we used in the earlier proof, and the $\tilde{p}_{0,0,n}$ bound of $\frac{6}{4n+2}$ is slightly weaker than the old bound of $\frac{1}{n}$.) A proof entirely similar to that of Lemma 7.2 now establishes that $\tilde{p}_{k,d,n} = 1/n^{\omega(1)}$ for $1 \leq k \leq (\log n)^{3/2}$ and $d \geq \frac{1}{2}\log n$, $d = \Theta(\log n)$. We now observe that the recurrence for $\tilde{p}_{k,d,n}$ dominates the recurrence which we have in this section for $p_{k,d,n}$, and hence $p_{k,d,n} \leq \tilde{p}_{k,d,n} = 1/n^{\omega(1)}$ as well. (Note that we cannot argue directly that the old $p_{k,d,n}$ recurrence dominates the new one, since some initial values of the new recurrence are as noted earlier slightly higher than the old values.) This proves Lemma E.2. ▯

It remains to prove Lemma E.3.

*Proof of Lemma* E.3. We have that

$$\Pr_{T \in \mathcal{T}^U_{d,n}}[S = 0 \mid x_1 \text{ is not a bottleneck}] = \frac{\Pr[S = 0 \text{ \& } x_1 \text{ is not a bottleneck}]}{\Pr[x_1 \text{ is not a bottleneck}]}$$
$$< 2\Pr[S = 0 \text{ \& } x_1 \text{ is not a bottleneck}]$$

since $\Pr[x_1 \text{ is not a bottleneck}] = 1 - p_{d,n} \geq \frac{1}{2}$ by Lemma E.1. Since $S = 0$ whenever $x_1$ is a bottleneck, we have that

$$\Pr[S = 0 \text{ \& } x_1 \text{ is not a bottleneck}] = \Pr[S = 0] - \Pr[x_1 \text{ is a bottleneck}].$$

Thus it suffices to bound $\epsilon_{d,n} \equiv p_{0,d,n} - p_{d,n}$. Combining (E.1) and (E.3) we have

$$\begin{aligned}
\epsilon_{d,n} &= (1 - q_{d,n}) + q_{d,n} \cdot \frac{n-1}{n}\left[(p_{0,d-1,n-1})^2 - (p_{d-1,n-1})^2\right] \\
&= (1 - q_{d,n}) + q_{d,n} \cdot \frac{n-1}{n}\left[(p_{0,d-1,n-1} - p_{d-1,n-1})(p_{0,d-1,n-1} + p_{d-1,n-1})\right] \\
&= (1 - q_{d,n}) + q_{d,n} \cdot \frac{n-1}{n}\left[\epsilon_{d-1,n-1}(p_{0,d-1,n-1} + p_{d-1,n-1})\right].
\end{aligned}$$

Our bounds on $p_{0,d,n}$ and $p_{d,n}$ imply that $p_{0,d-1,n-1} + p_{d-1,n-1} \leq \frac{5}{n-1}$. The above thus implies

$$\epsilon_{d,n} \leq (1 - q_{d,n}) + \epsilon_{d-1,n-1} \cdot \frac{5}{n-1}.$$

It is clear that for $\ell \geq \frac{1}{4}\log n$, we have $1-q_{\ell,n} = \frac{2}{C_{\ell,n}} = 1/2^{n^{\Omega(1)}}$. Since $\epsilon_{\frac{1}{4}\log n, n-d+\frac{1}{4}\log n}$ is clearly at most 1, expanding out the above recurrence we thus have

$$\epsilon_{d,n} < \frac{d-\frac{1}{4}\log n}{2^{n^{\Omega(1)}}} + \left(\frac{10}{n}\right)^{d-\frac{1}{4}\log n}.$$

Since $d \geq \frac{1}{2}\log n, d = \Theta(\log n)$ this is $\frac{1}{n^{\omega(1)}}$, and the lemma is proved. ◻

Finally, we can also now prove two lemmas used in section 5.

*Proof of Lemma* 5.7. It is easy to see that $C_{t,\Omega(n)} = 2^{n^{\Omega(1)}}$ for $t \geq \frac{1}{4}\log n$. Also, there are only poly($n$) chances for `MakeTree` to output a trivial tree down to depth $d - \frac{1}{4}\log n$. Therefore, the chance of any leaf being output before depth $\frac{1}{4}\log n$ is at most $2^{n^{\Omega(1)}}$. ◻

*Proof of Lemma* 5.5. In all three models we have that if $d = O(\log n)$, then $p_{d,n} < 2/n$. If $T$ is drawn according to $\mathcal{T}_{d,n}$ and has a bottleneck variable $x_1$ labeling node $v$ at depth $k \geq \frac{1}{8}\log n$, then the sibling of $v$ and the sibling of each of $v$'s nonroot ancestors must have $x_1$ as a bottleneck. These probabilities are all independent, and each is at most $(2/n)$. ◻

## REFERENCES

[1] A. Blum, *Rank-r decision trees are a subclass of r-decision lists*, Inform. Process. Lett., 42 (1992), pp. 83–185.

[2] A. Blum, M. Furst, M. Kearns, and R. Lipton, *Cryptographic primitives based on hard learning problems*, in Advances in Cryptology—CRYPTO '93, Lecture Notes in Comput. Sci. 773, Springer, Berlin, 1994, pp. 278–291.

[3] A. Blum, M. Furst, J. Jackson, M. Kearns, Y. Mansour, and S. Rudich, *Weakly learning DNF and characterizing statistical query learning using Fourier analysis*, in Proceedings of the 26th Annual ACM Symposium on Theory of Computing, ACM, New York, 1994, pp. 253–262.

[4] L. Breiman, J. Friedman, R. Olshen, and C. Stone, *Classification of Regression Trees*, Wadsworth, Belmont, CA, 1984.

[5] N. Bshouty, *Exact learning boolean functions via the monotone theory*, Inform. and Comput., 123 (1995), pp. 146–153.

[6] A. Ehrenfeucht and D. Haussler, *Learning decision trees from random examples*, Inform. and Comput., 82 (1989), pp. 231–246.

[7] T. Hancock, *Learning kμ decision trees on the uniform distribution*, in Proceedings of the 6th Annual ACM Conference on Computational Learning Theory, ACM, New York, 1993, pp. 352–360.

[8] J. Jackson and R. Servedio, *Learning random log-depth decision trees under the uniform distribution*, in Proceedings of the 16th Annual Conference on Computational Learning Theory, Lecture Notes in Comput. Sci. 2777, Springer, Berlin, 2003, pp. 610–624.

[9] E. Kushilevitz and Y. Mansour, *Learning decision trees using the Fourier spectrum*, SIAM J. Comput., 22 (1993), pp. 1331–1348.

[10] J. Quinlan, *C4.5: Programs for Machine Learning*, Morgan–Kaufmann, San Mateo, CA, 1993.

SIAM J. COMPUT.
Vol. 34, No. 5, pp. 1129–1146

# A NEW MULTILAYERED PCP AND THE HARDNESS OF HYPERGRAPH VERTEX COVER[*]

IRIT DINUR[†], VENKATESAN GURUSWAMI[‡], SUBHASH KHOT[§], AND ODED REGEV[¶]

**Abstract.** Given a $k$-uniform hypergraph, the E$k$-Vertex-Cover problem is to find the smallest subset of vertices that intersects every hyperedge. We present a new multilayered probabilistically checkable proof (PCP) construction that extends the Raz verifier. This enables us to prove that E$k$-Vertex-Cover is NP-hard to approximate within a factor of $(k-1-\varepsilon)$ for arbitrary constants $\varepsilon > 0$ and $k \geq 3$. The result is nearly tight as this problem can be easily approximated within factor $k$. Our construction makes use of the biased long-code and is analyzed using combinatorial properties of $s$-wise $t$-intersecting families of subsets.

We also give a different proof that shows an inapproximability factor of $\lfloor \frac{k}{2} \rfloor - \varepsilon$. In addition to being simpler, this proof also works for superconstant values of $k$ up to $(\log N)^{1/c}$, where $c > 1$ is a fixed constant and $N$ is the number of hyperedges.

**Key words.** hypergraph vertex cover, hardness of approximation, probabilistically checkable proof, multilayered outer verifier, long-code

**AMS subject classifications.** 68Q15, 68Q17

**DOI.** 10.1137/S0097539704443057

## 1. Introduction.

A $k$-uniform hypergraph $H = (V, E)$ consists of a set of vertices $V$ and a collection $E$ of $k$-element subsets of $V$ called hyperedges. A *vertex cover* of $H$ is a subset $S \subseteq V$ such that every hyperedge in $E$ intersects $S$; i.e., $e \cap S \neq \emptyset$ for each $e \in E$. An *independent set* in $G$ is a subset whose complement is a vertex cover or, in other words, a subset of vertices that contains no hyperedge entirely within it. The E$k$-Vertex-Cover problem is the problem of finding a minimum size vertex cover in a $k$-uniform hypergraph. This problem is alternatively called the minimum hitting set problem with sets of size $k$ (and is equivalent to the set cover problem where each element of the universe occurs in exactly $k$ sets).

The E$k$-Vertex-Cover problem is a fundamental NP-hard optimization problem. For $k = 2$, it is just the famous vertex cover problem on graphs. Owing to its NP-hardness, one is interested in how well it can be approximated in polynomial time. A very simple algorithm that is often taught in an undergraduate algorithms class is the following: greedily pick a maximal set of pairwise disjoint hyperedges and then

[*]Received by the editors April 14, 2004; accepted for publication (in revised form) December 3, 2004; published electronically June 3, 2005. A preliminary version of this paper appeared in *Proceedings of the 35th Annual ACM Symposium on Theory of Computing (STOC)*, San Diego, CA, 2003.

http://www.siam.org/journals/sicomp/34-5/44305.html

[†]The Miller Institute, University of California at Berkeley, Berkeley, CA 94720 (iritd@cs.berkeley.edu).

[‡]Department of Computer Science, University of Washington, Seattle, WA 98195 (venkat@cs.washington.edu). Part of this work was done while the author was at UC Berkeley as a Miller Research Fellow.

[§]Institute for Advanced Study, Princeton, NJ 08544 and Georgia Institute of Technology, Atlanta, GA 30332 (khot@math.ias.edu). This work was done while the author was at Princeton University, supported by Prof. Sanjeev Arora's David and Lucile Packard Fellowship, NSF grant CCR-0098180, and an NSF ITR grant.

[¶]Department of Computer Science, Tel-Aviv University, Tel-Aviv 69978, Israel (odedr@cs.tau.ac.il). Most of this work was done while the author was at the Institute for Advanced Study, Princeton, NJ, and was supported by NSF grant CCR-9987845.

include all vertices in the chosen hyperedges in the vertex cover. It is easy to show that this gives a factor $k$ approximation algorithm for E$k$-Vertex-Cover. State-of-the-art techniques yield only a tiny improvement, achieving a $k - o(1)$ approximation ratio [14]. This raises the question of whether achieving an approximation factor of $k - \varepsilon$ for any constant $\varepsilon > 0$ could be NP-hard. In this paper, we prove a nearly tight hardness result for E$k$-Vertex-Cover, as described in the following theorem.

THEOREM 1.1 (main theorem). *For every integer $k \geq 3$ and every $\varepsilon > 0$, it is NP-hard to approximate Ek-Vertex-Cover within a factor of $(k - 1 - \varepsilon)$.*

**Previous hardness results.** The vertex cover problem on hypergraphs where the size of the hyperedges is unbounded is nothing but the set cover problem. For this problem there is a $\ln n$ approximation algorithm [23, 20] and a matching hardness factor of $(1 - o(1)) \ln n$ due to Feige [10]. Feige showed that an approximation algorithm achieving a factor of $(1 - o(1)) \ln n$ would imply NP $\subseteq$ DTIME($n^{O(\log \log n)}$), where $n$ is the number of hyperedges. The best NP-hardness result (where the reduction is polynomial time) is due to Raz and Safra [25], who showed that it is NP-hard to approximate set cover to within $c \cdot \log n$ for some small $c > 0$. The first explicit hardness result shown for E$k$-Vertex-Cover was due to Trevisan [27], who considered the approximability of bounded degree instances of several combinatorial problems and specifically showed an inapproximability factor of $\Omega(k^{1/19})$ for E$k$-Vertex-Cover. Holmerin [18] showed that E4-Vertex-Cover is NP-hard to approximate within $(2-\varepsilon)$. Independently, Goldreich [12] showed a direct "FGLSS"-type [11] reduction (involving no use of the long-code, a crucial component in most recent probabilistically checkable proof (PCP) constructions) attaining a hardness factor of $(2-\varepsilon)$ for E$k$-Vertex-Cover for some constant $k$. Later, Holmerin [19] showed that E$k$-Vertex-Cover is NP-hard to approximate within a factor of $\Omega(k^{1-\varepsilon})$, and also that it is NP-hard to approximate E3-Vertex-Cover within factor $(3/2 - \varepsilon)$.

More recently Dinur, Guruswami, and Khot [7] gave a fairly simple proof of an $\alpha \cdot k$ hardness result for E$k$-Vertex-Cover for some $\alpha > \frac{1}{3}$. The proof takes a combinatorial view of Holmerin's construction and instead of Fourier analysis uses some properties concerning intersecting families of finite sets. The authors also give a more complicated reduction that shows a factor $(k - 3 - \varepsilon)$ hardness for E$k$-Vertex-Cover. The crucial impetus for that work came from the recent result of Dinur and Safra [9] on the hardness of approximating vertex cover (on graphs), and, as in [9], the notion of biased long-codes and some extremal combinatorics relating to intersecting families of sets play an important role. In addition to ideas from [9], the factor $(k - 3 - \varepsilon)$ hardness result also exploits the notion of covering complexity introduced by Guruswami, Håstad, and Sudan [13]. Neither the $\alpha \cdot k$ result nor the $k - 3 - \varepsilon$ result has been published (an ECCC manuscript exists [7]) since they have been subsumed by the work presented herein.

**Our result and techniques.** In this paper we improve upon all the above hardness results by proving a factor $(k-1-\varepsilon)$ inapproximability result for E$k$-Vertex-Cover. Already for $k = 3$, this is an improvement from $3/2 - \varepsilon$ to $2 - \varepsilon$. Improving our hardness factor from $(k - 1 - \varepsilon)$ to $(k - \varepsilon)$ appears highly nontrivial (although it was recently proved under a certain conjecture [22]). Note that such a bound would imply a factor $2 - \varepsilon$ hardness for vertex cover on graphs, a problem that is notoriously difficult. While our proof shares some of the extremal combinatorics flavor of [9] and [7], it draws its strength mainly from a new multilayered outer verifier system for NP languages. This multilayered system is constructed using the Raz verifier [24] as a building block.

The Raz verifier, which serves as the starting point or "outer verifier" in most (if not all) recent hardness results, can be described as follows. There are two sets of (non-Boolean) variables $Y$ and $Z$, and for certain pairs of $y \in Y$ and $z \in Z$, a constraint $\pi_{y\to z}$. The constraints are projections; i.e., for each assignment to $y$ there exists exactly one assignment to $z$ such that the constraint $\pi_{y\to z}$ is satisfied. The goal is to find an assignment $A$ to the variables so that a maximum number of constraints $\pi_{y\to z}$ are satisfied, i.e., have the property $\pi_{y\to z}(A(y)) = A(z)$. By the PCP theorem [2, 1] together with the parallel repetition theorem [24], we know that for every $\varepsilon > 0$ it is NP-hard to distinguish between the case where all the constraints can be satisfied and the case where no more than a fraction $\varepsilon$ of the constraints can be satisfied.

In [7], the $\alpha \cdot k$ hardness result is obtained by replacing every $Y$-variable by a block of vertices (representing its long-code). Hyperedges connect $y_1$-vertices to $y_2$-vertices only if there is some $z \in Z$ such that $\pi_{y_1\to z}, \pi_{y_2\to z}$ are constraints in the system. This construction has an inherent symmetry between blocks which deteriorates the projection property of the constraints, limiting the hardness factor one can prove to at most $k/2$. Since this is a relatively simple reduction, we include a proof showing a hardness of approximation factor of $(\lfloor k/2 \rfloor - \varepsilon)$. The improvement over the factor $k/3$ in [7] is obtained using a better result on $s$-wise $t$-intersecting set families. We also remark that this result itself suffices for the recent reduction from E$k$-Vertex-Cover to asymmetric $K$-center [5]. Moreover, this result has the advantage of working for much larger superconstant values of $k$ (up to $(\log N)^{1/c}$ for some absolute constant $c$, where $N$ is the number of hyperedges).

Another way of reducing the Raz verifier to E$k$-Vertex-Cover is by maintaining the asymmetry between $Y$ and $Z$, introducing a block of vertices for each variable in $Y$ and in $Z$ (representing their long-code). Each constraint $\pi_{y\to z}$ can be emulated by a set of hyperedges, where each hyperedge consists of both $y$-vertices and $z$-vertices. The hyperedges can be chosen so that if the initial PCP instance were satisfiable, then taking a particular fraction $1/k$ of the vertices in each block would be a vertex cover. However, this reduction has a basic "bipartiteness" flaw: the underlying constraint graph, being bipartite with parts $Y$ and $Z$, has a vertex cover of size at most one-half of the number of vertices. Taking all the vertices of, say, the $Z$-variables will be a vertex cover for the hypergraph regardless of whether or not the initial PCP instance was satisfiable. This, once again, limits the gap to no more than $k/2$.

We remark that this "bipartiteness" flaw naturally arises in other settings as well. One example is approximate hypergraph coloring, where indeed our multilayered PCP construction has been successfully used for showing hardness; see [8, 21].

*The multilayered PCP.* We overcome the $k/2$ limit by presenting a new, multilayered PCP. In this construction we maintain the projection property of the constraints that is a strong feature of the Raz verifier, while overcoming the "bipartiteness" flaw. In the usual Raz verifier we have two "layers," the first containing the $Y$-variables and the second containing the $Z$-variables. In the multilayered PCP, we have $\ell$ layers containing variables $X_1, X_2, \dots, X_\ell$, respectively. Between every pair of layers $i_1$ and $i_2$, we have a set of projection constraints that represent an instance of the Raz verifier. In the multilayered PCP, it is NP-hard to distinguish between (i) the case where there exists an assignment that satisfies *all* the constraints, between *every* pair of layers, and (ii) the case where for *every* $i_1, i_2$ it is impossible to satisfy more than a fraction $\varepsilon$ of the constraints between $X_{i_1}$ and $X_{i_2}$.

In addition, we prove that the underlying constraint graph no longer has the "bipartiteness" obstacle; i.e., it no longer has a small vertex cover, and hence it has no large independent set. Indeed we show that the multilayered PCP has a certain "weak-density" property: for any set containing an $\varepsilon$ fraction of the variables there are many constraints between variables of this set. This guarantees that "fake" independent sets in the hypergraph (i.e., independent sets that occur because there are no constraints between the variables of the set) contain at most an $\varepsilon$ fraction of the vertices.

We mention that the PCP presented by Feige in [10] has a few structural similarities with ours. Most notably, both have more than two types of variables. However, while in our construction the types are layered with decreasing domain sizes, in Feige's construction the different types are all symmetric. Furthermore, and more importantly, the constraints tested by the verifier in Feige's construction are not projections, while this is a key feature of our multilayered PCP, crucially exploited in our analysis.

We view the construction of the multilayered PCP as a central contribution of our paper and believe that it could be a powerful starting point for other hardness of approximation results as well. In fact, as mentioned above, our multilayered construction has already been used in obtaining strong hardness results for coloring 3-uniform hypergraphs [8, 21] (namely, the hardness of coloring a 2-colorable 3-uniform hypergraph using an arbitrary constant number of colors), a problem for which no nontrivial inapproximability results are known using other techniques. We anticipate that this new outer verifier will also find other applications besides the ones in this paper and in [8, 21].

*The biased long-code.* Our hypergraph construction relies on the long-code that was introduced in [3] and, more specifically, on the biased long-code defined in [9]. Thus, each PCP-variable is represented by a block of vertices, one for each "bit" of the biased long-code. More specifically, in $x$'s block we have one vertex for each subset of $R$, where $R$ is the set of assignments for the variable $x$. However, rather than taking all vertices in a block with equal weight, we attach weights to the vertices according to the $p$-biased long-code. The weight of a subset $F$ is set to $p^{|F|}(1-p)^{|R\setminus F|}$, highlighting subsets of cardinality $p \cdot |R|$. Thus we actually construct a weighted hypergraph; we will later describe how it can be translated to a nonweighted one.

The vertex cover in the hypergraph is shown to have relative size of either $1-p$ in the good case or almost 1 in the bad case. Choosing large $p = 1 - \frac{1}{k-1-\varepsilon}$ yields the desired gap of $\frac{1}{1-p} \approx k-1-\varepsilon$ between the good and bad cases. The reduction uses the following combinatorial property: *a family of subsets of a set $R$, where each subset has size $p\,|R|$, either contains very few subsets, or it contains some $k-1$ subsets whose common intersection is very small.* We will later show that this property holds for $p < 1 - \frac{1}{k-1}$, and hence we obtain a gap of $k-1-\varepsilon$. As can be seen, this property does not hold for $p > 1 - \frac{1}{k-1}$, and hence one cannot improve the $k-1-\varepsilon$ result by simply increasing $p$.

**Weighted versus unweighted.** As mentioned above, our construction yields a weighted hypergraph: each vertex is associated with a weight, and the goal is to minimize the weight of the vertex cover. By appropriately duplicating vertices, we can obtain an unweighted hypergraph (for more detail, see [6, 9]). We note that in all of our constructions, duplicating vertices does not change the asymptotic size of the hypergraph, and hence all of our results carry over to the unweighted case.

**Location of the gap.** All of our hardness results have the gap between sizes of the vertex cover at the "strongest" location. Specifically, to prove a factor $(k-1-\varepsilon)$

hardness we show that it is hard to distinguish between $k$-uniform hypergraphs that have a vertex cover of weight $\frac{1}{k-1} + \varepsilon$ and those whose minimum vertex cover has weight at least $(1 - \varepsilon)$. This result is stronger than a gap of about $(k - 1)$ achieved, for example, between vertex covers of weight $\frac{1}{(k-1)^2}$ and $\frac{1}{k-1}$. In fact, by adding dummy vertices, our result implies that for any $c < 1$ it is NP-hard to distinguish between hypergraphs whose minimum vertex cover has weight at least $c$ and those which have a vertex cover of weight at most $\left(\frac{c}{k-1} + \varepsilon\right)$. Put another way, our result shows that for any $k \geq 3$ there exists an $\alpha = \alpha(k) > 0$ such that for arbitrarily small $\varepsilon > 0$, it is NP-hard to find an independent set consisting of a fraction $\varepsilon$ of the vertices in a given $k$-uniform hypergraph, even if the hypergraph is promised to contain an independent set comprising a fraction $\alpha$ of the vertices. We remark that such a result is not known for graphs and seems out of the reach of current techniques. (The 1.36 hardness result for vertex cover on graphs due to Dinur and Safra [9], for example, shows that it is NP-hard to distinguish between cases when the graph has an independent set of size $0.38 \cdot n$ and cases when no independent set has more than $0.16 \cdot n$ vertices.) This gap location feature was crucial to the recent tight $\Omega(\log^* n)$ inapproximability result for asymmetric $K$-center [5].

**Organization.** We begin in section 2.1 by developing the machinery from extremal combinatorics concerning intersecting families of sets that will play a crucial role in our proof. This is followed by a statement of the starting point of our reduction, which is the standard gap instance obtained by combining the PCP theorem with Raz's parallel repetition theorem. In section 3, we present a relatively simple reduction that shows an inapproximability factor of $\lfloor k/2 \rfloor - \varepsilon$ for arbitrary $\varepsilon > 0$. As explained earlier, this is as far as we can go using just the "bipartite" Raz outer verifier. In section 4, we present the multilayered PCP construction, which offers hope of breaking the $k/2$ barrier. In section 5, we present our reduction to a gap version of E$k$-Vertex-Cover which allows us to prove a factor $(k-1-\varepsilon)$ inapproximability result for this problem. Finally, we discuss the case when $k$ is not a constant but instead is a growing function of the number of hyperedges in section 6.

## 2. Preliminaries.

### 2.1. Intersecting families.

In this section we define the notion of an $s$-wise $t$-intersecting family and prove an important property of such families. For a comprehensive survey, see [15]. Denote $[n] = \{1, \ldots, n\}$ and $2^{[n]} = \{F \mid F \subseteq [n]\}$. We start with the following definition.

Definition 2.1. *A family $\mathcal{F} \subseteq 2^{[n]}$ is called* $s$-wise $t$-intersecting *if for every $s$ sets $F_1, \ldots, F_s \in \mathcal{F}$, we have $|F_1 \cap \cdots \cap F_s| \geq t$.*

For example, the family of all sets $F \in 2^{[n]}$ such that $[t] \subseteq F$ is an $s$-wise $t$-intersecting family. A more interesting example is given by the family of all sets $F \in 2^{[n]}$ such that $|F \cap [t+s]| \geq t + s - 1$. Each set in this family has at most one "hole" in $[t+s]$. Therefore, any intersection of $s$ sets of this family contains at least $t + s - s = t$ elements from the range $[t+s]$. More generally, for any $j \geq 0$, we can define the family of all sets $F \in 2^{[n]}$ such that $|F \cap [t + js]| \geq t + (s-1)j$; it is easy to see that this is an $s$-wise $t$-intersecting family.

We need another important definition.

Definition 2.2. *For a bias parameter $0 < p < 1$ and a ground set $R$, the weight of a set $F \subseteq R$ is*

$$\mu_p^R(F) \overset{def}{=} p^{|F|} \cdot (1-p)^{|R \setminus F|}.$$

*When $R$ is clear from the context we write $\mu_p$ for $\mu_p^R$. The weight of a family $\mathcal{F} \subseteq 2^R$ is $\mu_p(\mathcal{F}) = \sum_{F \in \mathcal{F}} \mu_p(F)$.*

The weight of a subset is precisely the probability of obtaining this subset when one picks every element in $R$ independently with probability $p$.

The following is the main lemma of this section. It shows that for any $p < \frac{s-1}{s}$, a family of nonnegligible $\mu_p$-weight (i.e., $\mu_p(\mathcal{F}) \geq \varepsilon$) cannot be $s$-wise $t$-intersecting for sufficiently large $t$.

LEMMA 2.3. *For arbitrary $\varepsilon, \delta > 0$, and integer $s \geq 2$, let $p = 1 - \frac{1}{s} - \delta$. Then, there exists $t = t(\varepsilon, s, \delta)$ such that for any $s$-wise $t$-intersecting family $\mathcal{F} \subseteq 2^{[n]}$, $\mu_p(\mathcal{F}) < \varepsilon$. Moreover, it is enough to choose $t(\varepsilon, s, \delta) = \Omega(\frac{1}{\delta^2}(\log \frac{1}{\varepsilon} + \log(1 + \frac{1}{s\delta^2})))$.*

*Proof.* In order to prove this lemma, we need to introduce the notion of a left-shifted family. Performing an $(i,j)$-shift on a family consists of replacing the element $j$ with the element $i$ in all sets $F \in \mathcal{F}$ such that $j \in F$, $i \notin F$, and $(F \setminus \{j\}) \cup \{i\} \notin \mathcal{F}$. A left-shifted family is a family which is invariant with respect to $(i,j)$-shifts for any $1 \leq i < j \leq n$. For any family $\mathcal{F}$, by iterating the $(i,j)$-shift for all $1 \leq i < j \leq n$ we eventually get a left-shifted family which we denote by $S(\mathcal{F})$. The following simple lemma summarizes the properties of the left-shift operation.

LEMMA 2.4 (see [15, Lemma 4.2, p. 1298]). *For any family $\mathcal{F} \subseteq 2^{[n]}$, there exists a one-to-one and onto mapping $\tau$ from $\mathcal{F}$ to $S(\mathcal{F})$ such that $|F| = |\tau(F)|$ for every $F \in \mathcal{F}$. In other words, left-shifting a family maintains its size and the size of the sets in the family. Moreover, if $\mathcal{F}$ is an $s$-wise $t$-intersecting family, then so is $S(\mathcal{F})$.*

It can be easily checked that the examples given after Definition 2.1 are all left-shifted. As the next lemma shows, those examples essentially represent all possible sets in any left-shifted $s$-wise $t$-intersecting family.

LEMMA 2.5 (see [15, Lemma 8.3, p. 1311]). *Let $\mathcal{F} \subseteq 2^{[n]}$ be a left-shifted $s$-wise $t$-intersecting family. Then, for every $F \in \mathcal{F}$, there exists a $j \geq 0$ with $|F \cap [t+sj]| \geq t + (s-1)j$.*

*Proof.* For completeness, we sketch the proof of the lemma. For two equally sized sets $A = \{a_1, \ldots, a_l\}$, $1 \leq a_1 < \cdots < a_l \leq n$, and $B = \{b_1, \ldots, b_l\}$, $1 \leq b_1 < \cdots < b_l \leq n$, we say that $A \preceq B$ if $a_i \leq b_i$ for all $i = 1, \ldots, l$. Then, we claim that for such $A, B$, if $[n] \setminus A \in \mathcal{F}$, then also $[n] \setminus B \in \mathcal{F}$. We prove this by induction: for $i = 0, \ldots, l$ define the set $F_i = [n] \setminus \{a_1, \ldots, a_i, b_{i+1}, \ldots, b_l\}$. Notice that $F_l = [n] \setminus A$ and therefore $F_l \in \mathcal{F}$. Next, we show that $F_i \in \mathcal{F}$ implies that $F_{i-1} \in \mathcal{F}$. If $a_i = b_i$, then the claim is obvious. Otherwise, $a_i < b_i$ and hence $b_i \in F_i$. Since $\mathcal{F}$ is left-shifted and $a_i \notin F_i$, it follows that $F_i \setminus \{b_i\} \cup \{a_i\} = F_{i-1}$ is in $\mathcal{F}$. This proves our claim since $F_0 = [n] \setminus B$.

Let us now complete the proof of the lemma. Assume on the contrary that there exists $F \in \mathcal{F}$ such that for all $j \geq 0$, $|F \cap [t+sj]| < t + (s-1)j$. Let $A = \{a_1, \ldots, a_l\}$ be such that $F = [n] \setminus A$ and assume that $a_1 < \cdots < a_l$. The above condition implies that $F$ contains at least $i$ "holes" in $[t + (i-1)s]$ and therefore $a_i \leq t + (i-1)s$. It also implies that $l \geq \lfloor (n-t)/s \rfloor + 1$.

For $k = 0, \ldots, s-1$, define the set $B_k = \{b_{k,1}, \ldots, b_{k,l}\}$ by $b_{k,i} = \min\{t + k + (i-1)s, n - (l-i)\}$. Since $a_i \leq t + (i-1)s$ and $a_i \leq n - (l-i)$ (just because $a_i < a_{i+1} < \cdots < a_l \leq n$ are all integers), we obtain that $a_i \leq b_{0,i}$. In other words, $A \preceq B_0$. Moreover, since $b_{k,i} \leq b_{k+1,i}$, we see that $A \preceq B_0 \preceq B_1 \preceq \cdots \preceq B_{s-1}$. Using the claim we proved before, we obtain that for all $k = 0, \ldots, s-1$, $[n] \setminus B_k \in \mathcal{F}$. But this is a contradiction since $([n] \setminus B_0) \cap \cdots \cap ([n] \setminus B_{s-1}) \subseteq [t-1]$ and in particular its size is less than $t$. ☐

We now complete the proof of Lemma 2.3. We follow the general outline of the proof of Theorem 8.4 in [15, p. 1311]. Let $\mathcal{F}$ be an $s$-wise $t$-intersecting family where

$t$ will be determined later. According to Lemma 2.4, $S(\mathcal{F})$ is also $s$-wise $t$-intersecting and $\mu_p(S(\mathcal{F})) = \mu_p(\mathcal{F})$. By Lemma 2.5, for every $F \in S(\mathcal{F})$ there exists a $j \geq 0$ such that $|F \cap [t+sj]| \geq t+(s-1)j$. We can therefore bound $\mu_p(S(\mathcal{F}))$ from above by the probability that such a $j$ exists for a random set chosen according to the distribution $\mu_p$. We now prove an upper bound on this probability, which will give the desired bound on $\mu_p(S(\mathcal{F}))$ and hence also on $\mu_p(\mathcal{F})$.

The Chernoff bound [4] says that for a sequence of $m$ independent random variables $X_1, \dots, X_m$ on $\{0,1\}$ such that for all $i$, $\Pr[X_i = 1] = p$ for some $p$,

$$\Pr\left[\sum X_i > (p+\tau)m\right] \leq e^{-2m\tau^2}.$$

Hence, for any $j \geq 0$, $\Pr[\ |F \cap [t+sj]| \geq t+(s-1)j\ ]$ is at most

$$\Pr[\ |F \cap [t+sj]| - p(t+sj) \geq \delta(t+sj)\ ] \leq e^{-2(t+sj)\delta^2}.$$

Summing over all $j \geq 0$ we get

$$\mu_p(S(\mathcal{F})) \leq \sum_{j\geq 0} e^{-2(t+sj)\delta^2} = \frac{e^{-2t\delta^2}}{(1-e^{-2s\delta^2})} \leq e^{-2t\delta^2}\left(1+\frac{1}{2s\delta^2}\right),$$

which is smaller than $\varepsilon$ for $t = \Omega(\frac{1}{\delta^2}(\log\frac{1}{\varepsilon} + \log(1+\frac{1}{s\delta^2})))$. $\square$

**2.2. The PCP theorem and the parallel repetition theorem.** As is the case with many inapproximability results (e.g., [3, 16, 17, 26]), we begin our reduction from the Raz verifier described next. Let $R$ be some parameter and let $\Psi$ be a collection of two-variable constraints, where the variables are of two types, denoted $Y$ and $Z$. Let $R_Y$ denote the range of the $Y$-variables and $R_Z$ the range of the $Z$-variables,[1] where $|R_Z| \leq |R_Y|$ and both are at most $R^{O(1)}$. Assume that each constraint $\pi \in \Psi$ depends on exactly one $y \in Y$ and one $z \in Z$ and furthermore, that for every value $a_y \in R_Y$ assigned to $y$ there is exactly one value $a_z \in R_Z$ to $z$ such that the constraint $\pi$ is satisfied. Therefore, we can write each constraint $\pi \in \Psi$ as a function from $R_Y$ to $R_Z$ and use notation $\pi_{y\to z} : R_Y \to R_Z$. Furthermore, we assume that the underlying constraint graph is biregular; i.e., every $Y$-variable appears in the same number of constraints in $\Psi$, and every $Z$-variable appears in the same number of constraints in $\Psi$. Both of these numbers are assumed to be at most $R^{O(1)}$.

The following theorem follows by combining the PCP theorem with Raz's parallel repetition theorem. The PCP given by this theorem will henceforth be called the Raz verifier.

THEOREM 2.6 (PCP theorem [1, 2] and Raz's parallel repetition theorem [24]). *Let $\Psi$ be as above. There exists a universal constant $\gamma > 0$ such that for every (large enough) constant $R$ it is NP-hard to distinguish between the following two cases:*

- *Yes. There is an assignment $A : Y \to R_Y$, $A : Z \to R_Z$ such that all $\pi \in \Psi$ are satisfied by $A$, i.e., for all $\pi_{y\to z} \in \Psi$, $\pi_{y\to z}(A(y)) = A(z)$.*
- *No. No assignment can satisfy more than a fraction $\frac{1}{R^\gamma}$ of the constraints in $\Psi$.*

*Remark.* The reduction, from 3SAT say, proving the above hardness can be assumed to run in time $n^{O(\log R)}$, where $n$ is the size of the 3SAT instance. Also recall that the constraint graph of $\Psi$ is biregular with $Y$ and $Z$ degrees $d_l$ and $d_r$, where both $d_l, d_r \leq R^{O(1)}$.

[1]Readers familiar with the Raz verifier may prefer to think concretely of $R_Y = [7^u]$ and $R_Z = [2^u]$ for some number $u$ of repetitions.

**3. A factor $k/2$ inapproximability result.** In this section, we prove the factor $(\lfloor k/2 \rfloor - \varepsilon)$ hardness result for E$k$-Vertex-Cover for $k \geq 4$. We will show this by proving a factor $(k/2 - \varepsilon)$ hardness for even $k \geq 4$ (the result for odd $k$ follows by a trivial reduction that adds a new vertex to every hyperedge of a hard instance of the $(k-1)$-uniform hypergraph).

Let $IS(G)$ denote the weight of vertices contained in the largest independent set of the hypergraph $G$.

THEOREM 3.1. *Let $k \geq 4$ be an arbitrary even integer. Then for every $\varepsilon > 0$, it is NP-hard to approximate Ek-Vertex-Cover within a factor of $(k/2 - \varepsilon)$. More specifically, for every $\varepsilon, \delta > 0$, it is NP-hard to distinguish, given a k-uniform hypergraph G, between the following two cases:*
- $IS(G) \geq 1 - \frac{2}{k} - \delta$.
- $IS(G) \leq \varepsilon$.

*Proof.* Start with a PCP instance, as given in Theorem 2.6, namely, a set of local constraints $\Psi$ over variables $Y \cup Z$, whose respective ranges are $R_Y, R_Z$. For parameters, we pick $t$ to be larger than $t(\frac{\varepsilon}{2}, \frac{k}{2}, \delta)$ in Lemma 2.3, say, $t = O(\frac{1}{\delta^3} \log \frac{1}{\varepsilon})$ with some large enough constant. Moreover, take $R > (\frac{2t^2}{\varepsilon})^{1/\gamma}$, where $\gamma > 0$ is the universal constant from Theorem 2.6. From $\Psi$, we now construct a (weighted) $k$-uniform hypergraph $G$ whose minimum vertex cover has weight $\approx 2/k$ or $\approx 1$ depending on whether $\Psi$ is satisfiable or not.

For a set $R$ we denote by $2^R$ the power set of $R$. The vertex set of $G$ is

$$V \overset{def}{=} Y \times 2^{R_Y};$$

i.e., for each $y \in Y$ we construct a block of vertices denoted $V[y] = \{y\} \times 2^{R_Y}$ corresponding to all possible subsets of $R_Y$. The weight of each vertex $\langle y, F \rangle \in V$ is

$$\Lambda(\langle y, F \rangle) \overset{def}{=} \frac{1}{|Y|} \cdot \mu_{1-\frac{2}{k}-\delta}(F).$$

The hyperedges are defined as follows. For every pair of local constraints $\pi_{y_1 \to z}, \pi_{y_2 \to z} \in \Psi$ sharing a common variable $z \in Z$, we add the hyperedge

$$\{\langle y_1, F_1^1 \rangle, \langle y_1, F_2^1 \rangle, \ldots, \langle y_1, F_{k/2}^1 \rangle\} \bigcup \{\langle y_2, F_1^2 \rangle, \langle y_2, F_2^2 \rangle, \ldots, \langle y_2, F_{k/2}^2 \rangle\}$$

if and only if there is no $r_1 \in \bigcap_{1 \leq j \leq k/2} F_j^1$ and $r_2 \in \bigcap_{1 \leq j \leq k/2} F_j^2$ such that $\pi_{y_1 \to z}(r_1) = \pi_{y_2 \to z}(r_2)$. Formally,

$$E \overset{def}{=} \bigcup_{\pi_{y_1 \to z}, \pi_{y_2 \to z} \in \Phi} \Big\{ \{\langle y_1, F_1^1 \rangle, \ldots, \langle y_1, F_{k/2}^1 \rangle, \langle y_2, F_1^2 \rangle, \ldots, \langle y_2, F_{k/2}^2 \rangle\} \Big| \quad \pi_{y_1 \to z}(F_1^1 \cap \cdots \cap F_{k/2}^1) \cap \pi_{y_2 \to z}(F_1^2 \cap \cdots \cap F_{k/2}^2) = \emptyset \Big\},$$

where the union is taken over all pairs of local constraints with a common variable $z$.

LEMMA 3.2 (completeness). *If $\Psi$ is satisfiable, then $IS(G) \geq 1 - \frac{2}{k} - \delta$.*

*Proof.* Assume a satisfying assignment $A : Y \cup Z \to R_Y \cup R_Z$ for $\Psi$. The following set is an independent set of $G$:

$$\mathcal{I} = \{\langle y, F \rangle \in V[y] \mid y \in Y,\ A(y) \in F\}. \qquad (1)$$

For every hyperedge $e = \{\langle y_1, F_1^1 \rangle, \ldots, \langle y_1, F_{k/2}^1 \rangle, \langle y_2, F_1^2 \rangle, \ldots, \langle y_2, F_{k/2}^2 \rangle\}$ either $A(y_1) \notin F_1^1 \cap \cdots \cap F_{k/2}^1$ or $A(y_2) \notin F_1^2 \cap \cdots \cap F_{k/2}^2$; otherwise since $A(y_1), A(y_2)$ agree on every common $Z$-variable, we have $\pi_{y_1\to z}(F^1_1 \cap \cdots \cap F^1_{k/2}) \cap \pi_{y_2\to z}(F^2_1 \cap \cdots \cap F^2_{k/2}) \neq \emptyset$, and $e$ would not have been a hyperedge.

Now note that the weight of the family $\{F \mid A(y) \in F\}$ with respect to the bias parameter $(1 - 2/k - \delta)$ is $(1 - 2/k - \delta)$. Hence the weight of the independent set $\mathcal{I}$ in (1) is $1 - 2/k - \delta$. $\square$

LEMMA 3.3 (soundness). *If $IS(G) \geq \varepsilon$, then there exists an assignment $A : Y \cup Z \to R_Y \cup R_Z$ that satisfies more than a fraction $1/R^\gamma$ of the constraints in $\Psi$.*

*Proof.* Let $S \subset V$ be such an independent set of weight at least $\varepsilon$. We consider the set $Y' \subseteq Y$ of all variables $y$ for which the weight (under $\Lambda$) of $S \cap V[y]$ in $V[y]$ is at least $\varepsilon/2$. A simple averaging argument shows that $|Y'| \geq |Y|\varepsilon/2$.

For each $y \in Y'$, define

$$\mathcal{F}_y = \{F \in 2^{R_Y} \mid \langle y, F\rangle \in S\}.$$

Thus we have for all $y \in Y'$, $\mu_{1-\frac{2}{k}-\delta}(\mathcal{F}_y) \geq \frac{\varepsilon}{2}$. By Lemma 2.3, applied with the choice $s = k/2$ and $\varepsilon/2$ in place of $\varepsilon$, there must exist $F^y_1, F^y_2, \ldots, F^y_{k/2} \in \mathcal{F}_y$ such that $|F^y_1 \cap F^y_2 \cap \cdots \cap F^y_{k/2}| \leq t$, where $t$ is as defined before. Let us denote by $B(y)$ the intersection $F^y_1 \cap F^y_2 \cap \cdots \cap F^y_{k/2}$; we will refer to $B(y)$ as the *core* of assignments for $y$. Intuitively, for any $y$ for which the independent set $S$ has large intersection with $V[y]$, we can decode a small collection of potential values from $R_Y$, specifically the core $B(y)$, that may be assigned to $y$.

We next translate these cores into an assignment satisfying more than a fraction $\frac{1}{R^\gamma}$ of the constraints in $\Psi$. Observe that for every $z \in Z$ and $y_1, y_2 \in Y'$, $y_1 \neq y_2$, such that $\pi_{y_1\to z}, \pi_{y_2\to z} \in \Psi$, we must have

$$\pi_{y_1\to z}(B(y_1)) \cap \pi_{y_2\to z}(B(y_2)) \neq \emptyset. \tag{2}$$

Indeed, otherwise the set $\{\langle y_1, F^{y_1}_1\rangle, \ldots, \langle y_1, F^{y_1}_{k/2}\rangle, \langle y_2, F^{y_2}_1\rangle, \ldots, \langle y_2, F^{y_2}_{k/2}\rangle\}$ would be a hyperedge whose vertices all lie in $S$, contradicting the assumption that $S$ is an independent set. Let us now slightly modify the definition of $B(y)$: for any $y$ such that $B(y)$ is empty, we add to $B(y)$ some arbitrary element of $R_Y$. Notice that (2) still holds; moreover, it also holds in the case that $y_1$ and $y_2$ are equal.

Let $Z' \subseteq Z$ denote the set of all $Z$-variables of the instance $\Psi$ that participate in some constraint with some $y \in Y'$. Formally, $Z' \overset{def}{=} \{z \mid \pi_{y\to z} \in \Psi \text{ for some } y \in Y'\}$. Associate each such $z \in Z'$ with an arbitrary $y \in Y'$ for which $\pi_{y\to z} \in \Psi$, and let $B(z) \overset{def}{=} \pi_{y\to z}(B(y)) \subseteq R_Z$. Now define a random assignment $A$ by independently selecting for each $y \in Y', z \in Z'$ a random value from $B(y), B(z)$, respectively. Assign the rest of the variables $(Y \setminus Y') \cup (Z \setminus Z')$ with any arbitrary value. To complete the proof, we prove

$$E_A\big[\big|\{\pi_{y\to z} \in \Psi \mid \pi_{y\to z} \text{ is satisfied by } A\}\big|\big] \geq \frac{\varepsilon}{2t^2} \cdot |\Psi|\,. \tag{3}$$

Here the expectation is taken over the choice of the random assignment $A$. We will show that for every $y \in Y'$, each constraint $\pi_{y\to z} \in \Psi$ is satisfied by $A$ with probability $\geq \frac{1}{t^2}$; thus the expected number of local constraints satisfied by $A$ is $\frac{|Y'|}{|Y|} \cdot \frac{1}{t^2}|\Psi| \geq \frac{\varepsilon}{2t^2} \cdot |\Psi|$ (because each $y \in Y$ appears in the same number of local constraints).

So, let $\pi_{y\to z} \in \Psi$ for arbitrary $y \in Y'$. Assume $z$ is associated with some $y' \in Y'$ possibly equal to $y$. By (2), we have

$$\pi_{y\to z}(B(y)) \cap B(z) \;=\; \pi_{y\to z}(B(y)) \cap \pi_{y'\to z}(B(y')) \;\neq \emptyset.$$

Therefore, there is at least one value $a_y \in B(y)$ such that $\pi_{y\to z}(a_y) \in B(z)$. Since for every $y \in Y'$, $|B(y)| \leq t$, there is at least a $\frac{1}{t^2}$ probability of having $\pi_{y\to z}(A(y)) = A(z)$, thereby showing (3).

Thus, there exists some assignment $A$ that meets the expectation, which means it satisfies at least $\frac{\varepsilon}{2t^2} > \frac{1}{R^\gamma}$ (by our choice of $R$) of the local constraints in $\Psi$. This completes the proof of the soundness Lemma 3.3. $\square$

Theorem 3.1 now follows from Lemmas 3.2 and 3.3 and Theorem 2.6. $\square$

Let us compute some parameters of the reduction, which will be useful in section 6 where we will consider the case with superconstant values of $k$. First, let us compute $R$. The condition $R \geq (2t^2/\varepsilon)^{1/\gamma}$, together with our choice of $t$, implies that it suffices if

$$R = \left(\frac{1}{\varepsilon\delta}\right)^{O(1)}.$$

By the remark following Theorem 2.6, the number of variables in the PCP instance is at most $n^{O(\log R)} = n^{O(\log(1/\varepsilon\delta))}$, where $n$ is the size of the original 3SAT instance. Since there is a block of $2^{R^{O(1)}}$ vertices corresponding to all subsets of $R_y$ for each of these variables, the number of vertices in the hypergraph produced by the reduction is at most

$$n^{O(\log(1/\varepsilon\delta))} 2^{R^{O(1)}} \leq n^{O(\log(1/\varepsilon\delta))} 2^{(1/\varepsilon\delta)^{O(1)}}. \tag{4}$$

The degree of the hypergraph, i.e., the maximum number of hyperedges in which each vertex appears, is at most $2^{R^{O(1)}k} \cdot R^{O(1)}$, since each variable in the PCP instance appears in at most $R^{O(1)}$ constraints. Hence, the degree is at most

$$2^{k/(\varepsilon\delta)^{O(1)}}.$$

Finally, the running time of the reduction is polynomial in the number of hyperedges. A bound on the latter can be obtained by combining the two previous bounds:

$$n^{O(\log(1/\varepsilon\delta))} 2^{(1/\varepsilon\delta)^{O(1)}} \cdot 2^{k/(\varepsilon\delta)^{O(1)}} \leq n^{O(\log(1/\varepsilon\delta))} 2^{k/(\varepsilon\delta)^{O(1)}}. \tag{5}$$

## 4. The multilayered PCP.

As discussed in the introduction, a natural approach to build a hypergraph from the PCP $\Psi$ is to have a block of vertices for every variable $y$ or $z$ and define hyperedges of the hypergraph so as to enforce the constraints $\pi_{y\to z}$. For every constraint $\pi_{y\to z}$, there will be hyperedges containing vertices from the block of $y$ and the block of $z$. However, this approach is limited by the fact that the constraint graph underlying the PCP has a small vertex cover. Since each hyperedge contains vertices from both the $Y$ and $Z$ "sides," the subset of all vertices on the $Y$ (resp., $Z$) "side" already covers all of the hyperedges regardless of whether the initial PCP system was satisfiable or not.

This difficulty motivates our construction of a multilayered PCP where we have many types of variables (rather than only $Y$ and $Z$) and the resulting hypergraph is *multipartite*. The multilayered PCP is able to maintain the properties of Theorem 2.6 between *every* pair of layers. Moreover, the underlying constraint graph has a special "weak-density" property that, roughly speaking, guarantees it will have only tiny independent sets (thus any vertex cover for it must contain almost all of the vertices).

**4.1. Layering the variables.** Let $\ell, R > 0$. Let us begin by defining an $\ell$-layered PCP. In an $\ell$-layered PCP there are $\ell$ sets of variables denoted by $X_1, \ldots, X_\ell$. The range of variables in $X_i$ is denoted $R_i$, with $|R_i| = R^{O(\ell)}$. For every $1 \le i < j \le \ell$ there is a set of constraints $\Phi_{ij}$ where each constraint $\pi \in \Phi_{ij}$ depends on exactly one $x \in X_i$ and one $x' \in X_j$. For any two variables we denote by $\pi_{x \to x'}$ the constraint between them if such a constraint exists. Moreover, the constraints in $\Phi_{ij}$ are projections from $x$ to $x'$; that is, for every assignment to $x$ there is exactly one assignment to $x'$ such that the constraint is satisfied.

In addition, as mentioned in the introduction, we would like to show a certain "weak-density" property of our multilayered PCP. This property is defined in the following definition.

DEFINITION 4.1. *An $\ell$-layered PCP is said to be* weakly-dense *if for any $\delta$, $2/\ell < \delta < 1$, given $m \ge \lceil \frac{2}{\delta} \rceil$ layers $i_1 < \cdots < i_m$, and given any sets $S_j \subseteq X_{i_j}$ for $j \in [m]$ such that $S_j \ge \delta |X_{i_j}|$, there always exist two sets $S_j$ and $S_{j'}$ such that the number of constraints between them is at least a $\frac{\delta^2}{4}$ fraction of the constraints between the layers $X_{i_j}$ and $X_{i_{j'}}$.*

THEOREM 4.2. *There exists a universal constant $\gamma > 0$ such that for any parameters $\ell, R$, there is a weakly-dense $\ell$-layered PCP $\Phi = \cup \Phi_{ij}$ such that it is NP-hard to distinguish between the following two cases:*

- *Yes. There exists an assignment that satisfies all the constraints.*
- *No. For every $i < j$, not more than $1/R^\gamma$ of the constraints in $\Phi_{ij}$ can be satisfied by an assignment.*

*Moreover, the theorem holds even if every variable participates in $R^{O(\ell)}$ constraints.*

*Proof.* Let $\Psi$ be a constraint system as in Theorem 2.6. We construct $\Phi = \cup \Phi_{ij}$ as follows. The variables $X_i$ of layer $i \in [\ell]$ are the elements of the set $Z^i \times Y^{\ell - i}$, i.e., all $\ell$-tuples where the first $i$ elements are $Z$-variables and the last $\ell - i$ elements are $Y$-variables. The variables in layer $i$ have assignments from the set $R_i = (R_Z)^i \times (R_Y)^{\ell - i}$ corresponding to an assignment to each variable of $\Psi$ in the $\ell$-tuple. It is easy to see that $|R_i| \le R^{O(\ell)}$ for any $i \in [\ell]$ and that the total number of variables is no more than $|\Psi|^{O(\ell)}$. For any $1 \le i < j \le \ell$ we define the constraints in $\Phi_{ij}$ as follows. A constraint exists between a variable $x_i \in X_i$ and a variable $x_j \in X_j$ if they contain the same $\Psi$ variables in the first $i$ and the last $\ell - j$ elements of their $\ell$-tuples. Moreover, for any $i < k \le j$ there should be a constraint in $\Psi$ between $x_{i,k}$ and $x_{j,k}$. More formally, denoting $x_i = (x_{i,1}, \ldots, x_{i,\ell})$ for $x_i \in X_i = Z^i \times Y^{\ell - i}$,

$$\Phi_{ij} = \left\{ \pi_{x_i, x_j} \;\middle|\; \begin{array}{l} x_i \in X_i, x_j \in X_j \\ \forall k \in [\ell] \setminus \{i+1, \ldots, j\}, x_{i,k} = x_{j,k}, \\ \forall k \in \{i+1, \ldots, j\}, \pi_{x_{i,k} \to x_{j,k}} \in \Psi \end{array} \right\}.$$

As promised, the constraints $\pi_{x_i \to x_j}$ are projections. Given an assignment $a = (a_1, \ldots, a_\ell) \in R_i$ to $x_i$, we define the consistent assignment $b = (b_1, \ldots, b_\ell) \in R_j$ to $x_j$ as $b_k = \pi_{x_{i,k} \to x_{j,k}}(a_k)$ for $k \in \{i+1, \ldots, j\}$ and $b_k = a_k$ for all other $k$.

In how many constraints does a variable $x$ participate? Let $x \in X_i$ for some $i \in [\ell]$. Then $x = (x_1, \ldots, x_\ell)$ has a constraint with $x' = (x'_1, \ldots, x'_\ell) \in X_j$ if $i < j$ and on each coordinate $k \in \{i+1, \ldots, j\}$, $\Psi$ contains a constraint between $x'_k$ and $x_k$. For each $k$ there are at most $R^{O(1)}$ constraints in $\Psi$ that touch $x_k$, so altogether there are $R^{O(j-i)}$ such constraints that touch $x$. This is similar for the case where

$j < i$, and summing these together there are at most $\ell \cdot R^{O(\ell)} = R^{O(\ell)}$ constraints that touch any given $x$.

The completeness of $\Phi$ follows easily from the completeness of $\Psi$. That is, assume we are given an assignment $A : Y \cup Z \to R_Y \cup R_Z$ that satisfies all the constraints of $\Psi$. Then, the assignment $B : \bigcup X_i \to \bigcup R_i$ defined by $B(x_1, \dots, x_\ell) = (A(x_1), \dots, A(x_\ell))$ is a satisfying assignment.

For the soundness part, assume that there exist two layers $i < j$ and an assignment $B$ that satisfies more than a $1/R^\gamma$ fraction of the constraints in $\Phi_{ij}$. We partition $X_i$ into classes such that two variables in $X_i$ are in the same class if and only if they are identical except possibly on coordinate $j$. The variables in $X_j$ are also partitioned according to coordinate $j$. Since more than $1/R^\gamma$ of the constraints in $\Phi_{ij}$ are satisfied, it must be the case that there exist a class $x_{i,1}, \dots, x_{i,j-1}, x_{i,j+1}, \dots, x_{i,\ell}$ in the partition of $X_i$ and a class $x_{j,1}, \dots, x_{j,j-1}, x_{j,j+1}, \dots, x_{j,\ell}$ in the partition of $X_j$ between which there exist constraints and the fraction of satisfied constraints is more than $1/R^\gamma$. We define an assignment to $\Psi$ as

$$A(y) = (B(x_{i,1}, \dots, x_{i,j-1}, y, x_{i,j+1}, \dots, x_{i,\ell}))_j$$

for $y \in Y$ and as

$$A(z) = (B(x_{j,1}, \dots, x_{j,j-1}, z, x_{j,j+1}, \dots, x_{j,\ell}))_j$$

for $z \in Z$. Notice that there is a one-to-one and onto correspondence between the constraints in $\Psi$ and the constraints between the two chosen classes in $\Phi$. Moreover, if the constraint in $\Phi$ is satisfied, then the constraint in $\Psi$ is also satisfied. Therefore, $A$ is an assignment to $\Psi$ that satisfies more than $1/R^\gamma$ of the constraints.

To prove that this multilayered PCP is *weakly-dense*, we recall the biregularity property mentioned above; i.e., each variable $y \in Y$ appears in the same number of constraints, and also each $z \in Z$ appears in the same number of constraints. Therefore, the distribution obtained by uniformly choosing a variable $y \in Y$ and then uniformly choosing one of the variables in $z \in Z$ with which it has a constraint is a uniform distribution on $Z$.

Take any $m = \lceil \frac{2}{\delta} \rceil$ layers $i_1 < \cdots < i_m$ and sets $S_j \subseteq X_{i_j}$ for $j \in [m]$ such that $S_j \geq \delta |X_{i_j}|$. Consider a random walk beginning from a uniformly chosen variable $x_1 \in X_1$ and proceeding to a variable $x_2 \in X_2$ chosen uniformly among the variables with which $x_1$ has a constraint. The random walk continues in a similar way to a variable $x_3 \in X_3$ chosen uniformly among the variables with which $x_2$ has a constraint and so on up to a variable in $X_\ell$. Denote by $E_j$ the indicator variable of the event that the random walk hits an $S_j$-variable when in layer $X_{i_j}$. From the uniformity of $\Psi$ it follows that for every $j$, $\Pr[E_j] \geq \delta$. Moreover, using the inclusion-exclusion principle, we get

$$\begin{aligned} 1 \geq \Pr\left[\bigvee E_j\right] &\geq \sum_j \Pr[E_j] - \sum_{j<k} \Pr[E_j \wedge E_k] \\ &\geq \left\lceil \frac{2}{\delta} \right\rceil \cdot \delta - \binom{m}{2} \max_{j<k} \Pr[E_j \wedge E_k] \\ &\geq 2 - \binom{m}{2} \max_{j<k} \Pr[E_j \wedge E_k], \end{aligned}$$

which implies

$$\max_{j<k} \Pr[E_j \wedge E_k] \geq 1 / \binom{m}{2} \geq \frac{\delta^2}{4}.$$

Fix $j$ and $k$ such that $\Pr[E_j \wedge E_k] \geq \frac{\delta^2}{4}$ and consider a shorter random walk beginning from a random variable in $X_{i_j}$ and proceeding to the next layer and so on until hitting layer $i_k$. Since $E_j$ is uniform on $X_{i_j}$ we still have that $\Pr[E_j \wedge E_k] \geq \frac{\delta^2}{4}$ where the probability is taken over the random walks between $X_{i_j}$ and $X_{i_k}$. Also, notice that there is a one-to-one and onto mapping from the set of all random walks between $X_{i_j}$ and $X_{i_k}$ to the set $\Phi_{i_j,i_k}$. Therefore, at least a fraction $\frac{\delta^2}{4}$ of the constraints between $X_{i_j}$ and $X_{i_k}$ are between $S_j$ and $S_k$, which completes the proof of the weak-density property. ▯

## 5. A factor $(k-1-\varepsilon)$ inapproximability result.

THEOREM 1.1 (main theorem). *For any $k \geq 3$, it is NP-hard to approximate Ek-Vertex-Cover within any factor less than $k-1$. More specifically, for every $\varepsilon, \delta > 0$, it is NP-hard to distinguish, given a $k$-uniform hypergraph $G$, between the following two cases:*

- *$IS(G) \geq 1 - \frac{1}{k-1} - \delta$.*
- *$IS(G) \leq \varepsilon$.*

*Proof.* Fix $k \geq 3$ and arbitrarily small $\varepsilon, \delta > 0$. Define $p = 1 - \frac{1}{k-1} - \delta$. Let $\Phi$ be a PCP instance with layers $X_1, \ldots, X_\ell$, as described in Theorem 4.2, with parameters $\ell = 33\varepsilon^{-2}$ and $R$ large enough to be chosen later. We present a construction of a $k$-uniform hypergraph $G = (V, E)$. We use the long-code introduced by Bellare, Goldreich, and Sudan [3]. A long-code over domain $R$ has one bit for every subset $v \subseteq R$. An encoding of element $x \in R$ assigns bit-value 1 to the sets $v$ such that $x \in v$ and assigns 0 to the sets which do not contain $x$. In the following, the bits in the long-code will be vertices of the hypergraph. The vertices that correspond to a bit-value 0 are (supposedly) the vertices of a vertex cover.

*Vertices.* For each variable $x$ in layer $X_i$ we construct a block of vertices $V[x]$. This block contains a vertex for each subset of $R_i$. Throughout this section we slightly abuse notation by writing a vertex rather than the set it represents. The weight of each vertex in the block $V[x]$ is set according to $\mu_p^{R_i}$; i.e., the weight of a subset $v \subseteq R_i$ is proportional to $\mu_p^{R_i}(v) = p^{|v|}(1-p)^{|R_i \setminus v|}$ as in Definition 2.2. All blocks in the same layer have the same total weight, and the total weight of each layer is $\frac{1}{\ell}$. Formally, the weight of a vertex $v \in V[x]$, where $x \in X_i$, is given by

$$\frac{1}{\ell|X_i|}\mu_p^{R_i}(v).$$

*Hyperedges.* We construct hyperedges between blocks $V[x]$ and $V[y]$ such that there exists a constraint $\pi_{x\to y}$. We connect a hyperedge between any $v_1, \ldots, v_{k-1} \in V[x]$ and $u \in V[y]$ whenever $\pi_{x\to y}(\bigcap_{i=1}^{k-1} v_i) \cap u = \emptyset$.

The intuition for the hyperedges comes from the proof of completeness. In fact, the hypergraph is constructed by first deciding how to map a satisfying assignment for $\Phi$ into a subset of $G$'s vertices. Then, we construct the hyperedges so as to ensure that such subsets are independent sets in $G$.

Let $IS(G)$ denote the weight of vertices contained in the largest independent set of the hypergraph $G$.

LEMMA 5.1 (completeness). *If $\Phi$ is satisfiable, then $IS(G) \geq p = 1 - \frac{1}{k-1} - \delta$.*

*Proof.* Let $A$ be a satisfying assignment for $\Phi$; i.e., $A$ maps each $i \in [\ell]$ and $x \in X_i$ to an assignment in $R_i$ such that all the constraints are satisfied. Let $\mathcal{I} \subseteq V$ contain in the block $V[x]$ all the vertices that contain the assignment $A(x)$:

$$\mathcal{I} = \bigcup_x \{v \in V[x] \mid v \ni A(x)\} \ .$$

We claim that $\mathcal{I}$ is an independent set. Take any $v_1, \ldots, v_{k-1}$ in $\mathcal{I} \cap V[x]$ and a vertex $u$ in $\mathcal{I} \cap V[y]$. The vertices $v_1, \ldots, v_{k-1}$ intersect on $A(x)$ and therefore the projection of their intersection contains $\pi_{x\to y}(A(x)) = A(y)$. Since $u$ is in $\mathcal{I} \cap V[y]$ it must contain $A(y)$. The proof is completed by noting that inside each block, the weight of the set of all vertices that contain a specific assignment is exactly $p$. ∎

We now turn to the soundness of the construction.

LEMMA 5.2 (soundness). *If $IS(G) \geq \varepsilon$, then $\Phi$ is satisfiable.*

*Proof.* Let $\mathcal{I}$ be an independent set of weight $\varepsilon$. We consider the set $X'$ of all variables $x$ for which the weight of $\mathcal{I} \cap V[x]$ in $V[x]$ is at least $\varepsilon/2$. A simple averaging argument shows that the weight of $\bigcup_{x\in X'} V[x]$ is at least $\frac{\varepsilon}{2}$. Another averaging argument shows that in at least $\frac{\varepsilon}{4}\ell > \frac{8}{\varepsilon}$ layers, $X'$ contains at least an $\frac{\varepsilon}{4}$ fraction of the variables. Using the weak-density property of the PCP (see Definition 4.1) with the choice $\delta = \varepsilon/4$, we conclude that there exist two layers $X_i$ and $X_j$ such that an $\frac{\varepsilon^2}{64}$ fraction of the constraints between them are constraints between variables in $X'$. Let us denote by $X$ the variables in $X_i \cap X'$ and by $Y$ the variables in $X_j \cap X'$.

We first find, for each variable $x \in X$, a short list $B(x)$ of values that are "assigned" to it by $\mathcal{I}$. Indeed consider the vertices in $\mathcal{I} \cap V[x]$. Define $t$ to be $O(\frac{1}{\delta^3} \log \frac{1}{\varepsilon})$ with some large enough constant. Then, since $t \geq t(\frac{\varepsilon}{2}, k-1, \delta)$, Lemma 2.3 implies that there exist $k-1$ vertices in $\mathcal{I} \cap V[x]$ that intersect in at most $t$ assignments. We denote these vertices by $v_{x,1}, \ldots, v_{x,k-1}$ and their intersection by $B(x)$, where $|B(x)| \leq t$.

In the following we define an assignment to the variables in $X$ and $Y$ such that many of the constraints between them are satisfied. For a variable $y \in Y$ we choose the assignment that is contained in the largest number of projections of $B(x)$:

$$A(y) = \underset{a\in R_Y}{\mathrm{maxvar}} \, |\{x \in X \mid a \in \pi_{x\to y}(B(x))\}|.$$

For a variable $x \in X$ we simply choose $A(x)$ to be a random assignment from the set $B(x)$ (or some arbitrary assignment in case $B(x)$ is empty). We will prove that indeed $A(y)$ occurs in many of the projections of $B(x)$.

CLAIM 5.3. *Let $y \in Y$, and let $X(y)$ be the set of all variables $x \in X$ that have a constraint with $y$. Then,*

$$\Pr_{x\in X(y)} [A(y) \in \pi_{x\to y}(B(x))] > \varepsilon_1 \overset{def}{=} \frac{1}{t \log\left(\frac{\varepsilon}{4}\right) / \log(1 - 1/(k-1)^t)} \ .$$

Before we prove the claim let us see how it concludes the proof. Indeed, the fraction of constraints between $Y$ and $X$ satisfied by $A$ is, in expectation, at least $\varepsilon_1/t$. This is because for every $y$, the fraction of $x \in X(y)$ for which $A(y) \in \pi_{x\to y}(B(x))$ is by Claim 5.3 at least $\varepsilon_1$. For such $x$, the probability that $A(x)$ was randomly chosen so that $\pi_{x\to y}(A(x)) = A(y)$ is at least $\frac{1}{|B(x)|} \geq \frac{1}{t}$.

Thus, there must be some assignment $A$ that attains the expectation and satisfies at least $\frac{\varepsilon_1}{t}$ of the constraints between $X$ and $Y$. This amounts to an $\frac{\varepsilon_1}{t} \cdot \frac{\varepsilon^2}{64}$ fraction of the total constraints, which would imply that $\Phi$ is satisfiable, as long as

$$\frac{1}{t \log(\frac{\varepsilon}{4}) / \log(1 - 1/(k-1)^t)} \cdot \frac{1}{t} \cdot \frac{\varepsilon^2}{64} \geq \frac{1}{R^\gamma} \ . \tag{6}$$

*Proof of Claim* 5.3. Let $y \in Y$, $x \in X(y)$. By definition, there are no hyperedges of the form $(v_{x,1}, \ldots, v_{x,k-1}, u)$ for any vertex $u \in \mathcal{I} \cap V[y]$. In other words, every vertex $u \in \mathcal{I} \cap V[y]$ must intersect $\pi_{x\to y}(B(x))$. Let $x_{i_1}, \ldots, x_{i_N}$ be the variables in $X$ such that the constraint $\pi_{x_{i_j}\to y}$ exists, and consider the collection of sets $A_j = \pi_{x_{i_j}\to y}(B(x_{i_j}))$ for $j = 1, \ldots, N$. Clearly $|A_j| \leq t$, and let $D$ denote the maximum number of disjoint sets inside this family. We will show that $D$ cannot be too large by estimating the maximum possible weight of the vertices in $\mathcal{I} \cap V[y]$. Indeed the vertices in $\mathcal{I} \cap V[y]$ must intersect every $A_j$. The constraint imposed by one $A_j$ in isolation reduces the maximum possible weight of the vertices in $\mathcal{I} \cap V[y]$ by a factor of $1 - (1-p)^{|A_j|} \leq 1 - (1-p)^t$. Moreover, for disjoint $A_j$'s, these constraints are independent, so the total weight becomes at most $(1 - (1-p)^t)^D$. Since the weight of $\mathcal{I} \cap V[y]$ is at least $\frac{\varepsilon}{4}$, we have

$$\frac{\varepsilon}{4} \leq (1 - (1-p)^t)^D < \left(1 - \frac{1}{(k-1)^t}\right)^D$$

so

$$D < \log\left(\frac{\varepsilon}{4}\right) / \log(1 - 1/(k-1)^t)\,.$$

To finish, we need the following simple claim.

Claim 5.4. *Let $A_1, \ldots, A_N$ be a collection of $N$ sets, each of size at most $T \geq 1$. If there are not more than $D$ pairwise disjoint sets in this collection, then there is an element that is contained in at least $\frac{N}{TD}$ sets.*

*Proof.* Take any maximal collection of pairwise disjoint sets and let $A'$ denote their union. Notice that $|A'| \leq TD$ and that any set not in this collection must contain one of the elements of $A'$. Hence, there must exist an element of $A'$ that is contained in at least $1 + \frac{N-D}{TD} \geq N/TD$ sets. □

Claim 5.4 implies that there exists an assignment for $y$ that is contained in at least

$$\frac{N}{t \cdot \log(\frac{\varepsilon}{4}) / \log(1 - 1/(k-1)^t)}$$

of the $A_j$'s. □

This completes the soundness proof (Lemma 5.2). □

Theorem 1.1 now follows from Lemmas 5.1 and 5.2, since the ratio between the sizes of the vertex cover in the yes and no cases is at least $\frac{1-\varepsilon}{1-p} = (1-\varepsilon)/(\frac{1}{k-1} + \delta)$, which can be made arbitrarily close to $k-1$. □

As before, let us compute some parameters of the reduction, which will be useful in the next section. We start by choosing $R$ so that (6) is satisfied. Since $\frac{1}{(k-1)^t}$ is small, we estimate $\log(1-x) \approx -x$ and get

$$R \geq \Omega\left(\frac{1}{\varepsilon^2} \log\frac{1}{\varepsilon} \cdot t^2 \cdot (k-1)^t\right)^{1/\gamma}. \tag{7}$$

By plugging in the value for $t$, we obtain that choosing

$$R = 2^{O(\delta^{-3} \log k \log \frac{1}{\varepsilon})}$$

suffices. The number of variables in the multilayered PCP is $n^{O(\ell \log R)}$, which, using $\ell = O(1/\varepsilon^2)$, is at most

$$n^{\left(\frac{\log k}{\varepsilon\delta}\right)^{O(1)}}.$$

Hence, the number of vertices in the hypergraph is at most

$$n^{\left(\frac{\log k}{\varepsilon\delta}\right)^{O(1)}} \cdot 2^{R^{O(l)}} \leq n^{\left(\frac{\log k}{\varepsilon\delta}\right)^{O(1)}} \cdot 2^{2^{\left(\frac{\log k}{\varepsilon\delta}\right)^{O(1)}}}.$$

Since the number of constraints in which each variable participates is at most $R^{O(\ell)}$, the degree of the hypergraph is at most

$$R^{O(\ell)} \cdot 2^{O(kR^{O(\ell)})} \leq 2^{2^{\left(\frac{\log k}{\varepsilon\delta}\right)^{O(1)}}}.$$

Finally, the running time of the reduction is polynomial in the number of hyperedges, which is at most

$$2^{2^{\left(\frac{\log k}{\varepsilon\delta}\right)^{O(1)}}} \cdot n^{\left(\frac{\log k}{\varepsilon\delta}\right)^{O(1)}}. \tag{8}$$

**6. Nonconstant values of *k*.** In this section we outline what happens to our construction when we increase $k$ beyond a constant. This should be contrasted with the general hypergraph vertex cover problem in which $k$ is unbounded (i.e., the set cover problem). It is well known that if $N$ is the number of hyperedges, the greedy algorithm achieves a factor $\ln N$ approximation, and no polynomial time algorithm can achieve an approximation factor of $(1-o(1))\ln N$, unless $\mathsf{NP} \subseteq \mathsf{DTIME}(n^{O(\log\log n)})$ [10].

Below, we discuss in turn the extension of both the $k-1$ and $k/2$ hardness bounds from sections 5 and 3, respectively, to the case when $k$ is superconstant. We note that in this context the $k/2$ result is not subsumed by the $(k-1)$ result since it works for much larger values of $k$ (as a function of the number of hyperedges in the hypergraph). We discuss the $(k-1)$ result first since this reduction and the calculations at the end of section 5 are, we hope, fresh in the reader's mind at this point of reading.

**6.1. The $(k-1-\varepsilon)$ inapproximability bound.**

THEOREM 6.1. *There exists some $c > 0$ such that unless $\mathsf{NP} \subseteq \mathsf{DTIME}(n^{O(\log\log n)})$, there is no polynomial time algorithm for approximating Ek-Vertex-Cover for $k \leq (\log\log N)^{1/c}$ to within a factor of $k-1.01$, where $N$ is the number of hyperedges in the $k$-uniform hypergraph.*

*Proof.* Recall the calculation at the end of section 5, specifically the bound on the number of hyperedges from (8). The produced hypergraph $G$ has

$$N = 2^{2^{\left(\frac{\log k}{\varepsilon\delta}\right)^{O(1)}}} \cdot n^{\left(\frac{\log k}{\varepsilon\delta}\right)^{O(1)}}$$

hyperedges, when starting from a 3SAT instance of size $n$. Furthermore, the reduction runs in time polynomial in the number of hyperedges.

The gap in vertex cover sizes equals

$$\frac{1-\varepsilon}{1/(k-1)+\delta} \geq (k-1)(1-\varepsilon)(1-k\delta) \geq (k-1.01)$$

if we set $\varepsilon = 1/(200k)$ and $\delta = 1/(200k^2)$. For these choices of $\varepsilon, \delta$, the number of hyperedges $N$ is at least $2^{2^{k^{O(1)}}}$, so the largest value of $k$ we can set is $k = (\log\log n)^{1/c'}$ for a large enough constant $c'$. For this choice of $k$, the number of hyperedges $N$ and the runtime of the reduction can both be bounded above by $n^{O(\log\log n)}$. For large enough $n$, $\log\log N \leq 2\log\log n$ for this choice of parameters, so we get a result that works for $k \leq (\log\log N)^{1/c}$ for some fixed constant $c$. $\square$

**6.2. The $(k/2-\varepsilon)$ inapproximability bound.** We can obtain a similar result corresponding to the factor $k/2$ hardness result, as stated below. Note that this result works for $k$ up to $(\log N)^{1/b}$ for some fixed constant $b > 0$.

THEOREM 6.2. *There exists some $b > 0$ such that unless $\mathsf{NP} \subseteq \mathsf{DTIME}(n^{O(\log\log n)})$, there is no polynomial time algorithm for approximating Ek-Vertex-Cover for $k \le (\log N)^{1/b}$ to within a factor of $\lfloor \frac{k}{2} \rfloor - 0.01$, where $N$ is the number of hyperedges in the $k$-uniform hypergraph.*

*Proof.* Recall the calculation at the end of section 3, specifically (4) and (5). The produced hypergraph $G$ has $n_0 \le n^{O(\log(1/\varepsilon\delta))} 2^{(1/\varepsilon\delta)^{O(1)}}$ vertices and $N = n_0 2^{k/(\varepsilon\delta)^{O(1)}}$ hyperedges, when starting from a 3SAT instance of size $n$. Furthermore, the reduction runs in time polynomial in the number of hyperedges.

By plugging in $k = (\log n)^{1/b'}$ for a large enough constant $b' > 0$, $\varepsilon = 1/(100k)$ and $\delta = 1/(100k^2)$, we get $\log(1/\varepsilon\delta) \le \log\log n$, and $2^{k/(\varepsilon\delta)^{O(1)}} \le n$. Thus, the reduction runs in time $n^{O(\log\log n)}$ and produces a hypergraph with $N = n^{O(\log\log n)}$ hyperedges, with a gap in vertex cover sizes (assuming $k$ is even for convenience) of

$$\frac{1-\varepsilon}{2/k+\delta} \ge \frac{k}{2}(1-\varepsilon)(1-k\delta) \ge \frac{k}{2} - \frac{1}{100}$$

by our choices of $\varepsilon, \delta$. Since for large enough $n$, $\log n \ge \sqrt{\log N}$ for the above choice of parameters, we get a result that works for $k \le (\log N)^{1/b}$ for some fixed constant $b$. ∎

**Acknowledgments.** We would like to thank Noga Alon for his help with $s$-wise $t$-intersecting families. We would also like to thank Luca Trevisan and the anonymous referees for useful comments.

## REFERENCES

[1] S. Arora, C. Lund, R. Motwani, M. Sudan, and M. Szegedy, *Proof verification and the hardness of approximation problems*, J. ACM, 45 (1998), pp. 501–555.

[2] S. Arora and S. Safra, *Probabilistic checking of proofs: A new characterization of NP*, J. ACM, 45 (1998), pp. 70–122.

[3] M. Bellare, O. Goldreich, and M. Sudan, *Free bits, PCPs, and nonapproximability—towards tight results*, SIAM J. Comput., 27 (1998), pp. 804–915.

[4] H. Chernoff, *A measure of asymptotic efficiency for tests of a hypothesis based on the sum of observations*, Ann. Math. Statistics, 23 (1952), pp. 493–507.

[5] J. Chuzhoy, S. Guha, E. Halperin, S. Khanna, G. Kortsarz, and J. Naor, *Asymmetric K-center is log* n hard to approximate*, in Proceedings of the 36th Annual ACM Symposium on Theory of Computing (STOC), 2004, pp. 21–27.

[6] P. Crescenzi, R. Silvestri, and L. Trevisan, *On weighted vs. unweighted versions of combinatorial optimization problems*, Inform. and Comput., 167 (2001), pp. 10–26.

[7] I. Dinur, V. Guruswami, and S. Khot, *Vertex Cover on k-Uniform Hypergraphs Is Hard to Approximate within Factor* $(k-3-\varepsilon)$, Tech. report TR02-027, Electronic Colloquium on Computational Complexity, 2002.

[8] I. Dinur, O. Regev, and C. Smyth, *The hardness of* 3*-uniform hypergraph coloring*, Combinatorica, to appear.

[9] I. Dinur and S. Safra, *The importance of being biased*, Ann. of Math., to appear.

[10] U. Feige, *A threshold of ln n for approximating set cover*, J. ACM, 45 (1998), pp. 634–652.

[11] U. Feige, S. Goldwasser, L. Lovász, S. Safra, and M. Szegedy, *Interactive proofs and the hardness of approximating cliques*, J. ACM, 43 (1996), pp. 268–292.

[12] O. Goldreich, *Using the FGLSS-Reduction to Prove Inapproximability Results for Minimum Vertex Cover in Hypergraphs*, Tech. report TR01-102, Electronic Colloquium on Computational Complexity, 2001.

[13] V. Guruswami, J. Håstad, and M. Sudan, *Hardness of approximate hypergraph coloring*, SIAM J. Comput., 31 (2002), pp. 1663–1686.

[14] E. Halperin, *Improved approximation algorithms for the vertex cover problem in graphs and hypergraphs*, SIAM J. Comput., 31 (2002), pp. 1608–1623.

[15] R. L. Graham, M. Grötschel, and L. Lovász, eds., *Handbook of Combinatorics,* Vols. 1 and 2, Elsevier Science, Amsterdam, 1995.

[16] J. Håstad, *Clique is hard to approximate within* $n^{1-\varepsilon}$, Acta Math. 182 (1999), pp. 105–142.

[17] J. Håstad, *Some optimal inapproximability results*, J. ACM, 48 (2001), pp. 798–859.

[18] J. Holmerin, *Vertex cover on* 4*-uniform hypergraphs is hard to approximate within* $2-\varepsilon$, in Proceedings of the 34th Annual ACM Symposium on Theory of Computing (STOC), 2002, pp. 544–552.

[19] J. Holmerin, *Improved inapproximability results for vertex cover on k-uniform hypergraphs*, in Automata, Languages and Programming, Lecture Notes in Comput. Sci. 2380, Springer-Verlag, Berlin, 2002, pp. 1005–1016.

[20] D. S. Johnson, *Approximation algorithms for combinatorial problems*, J. Comput. System Sci., 9 (1974), pp. 256–278.

[21] S. Khot, *Hardness results for coloring* 3*-colorable* 3*-uniform hypergraphs*, in Proceedings of the 43rd Annual IEEE Symposium on Foundations of Computer Science, 2002, pp. 23–32.

[22] S. Khot and O. Regev, *Vertex cover might be hard to approximate to within* $2-\varepsilon$, in Proceedings of 18th IEEE Annual Conference on Computational Complexity (CCC), 2003, pp. 379–386.

[23] L. Lovász, *On the ratio of optimal integral and fractional covers*, Discrete Math., 13 (1975), pp. 383–390.

[24] R. Raz, *A parallel repetition theorem*, SIAM J. Comput., 27 (1998), pp. 763–803.

[25] R. Raz and S. Safra, *A sub-constant error-probability low-degree test, and a sub-constant error-probability PCP characterization of NP*, in Proceedings of the 29th Annual ACM Symposium on Theory of Computing, 1997, pp. 475–484.

[26] A. Samorodnitsky and L. Trevisan, *A PCP characterization of NP with optimal amortized query complexity*, in Proceedings of the 32nd Annual ACM Symposium on Theory of Computing, 2000, pp. 191–199.

[27] L. Trevisan, *Non-approximability results for optimization problems on bounded degree instances*, in Proceedings of the 33rd Annual ACM Symposium on Theory of Computing, 2001, pp. 453–461.

SIAM J. COMPUT. © 2005 Society for Industrial and Applied Mathematics
Vol. 34, No. 5, pp. 1147–1158

# THE DIFFICULTY OF TESTING FOR ISOMORPHISM AGAINST A GRAPH THAT IS GIVEN IN ADVANCE[*]

ELDAR FISCHER[†]

**Abstract.** Motivated by a question from [E. Fischer, G. Kindler, D. Ron, S. Safra, and A. Samorodnitsky, *J. Comput. System Sci.*, 68 (2004), pp. 753–787], we investigate the number of queries required for testing that an input graph $G$ is isomorphic to a fixed graph $H$ that is given in advance. We correlate this number with a measure of the "complexity" of $H$ that we define here, by proving both an upper bound and a lower bound on the number of queries that depends on this new measure. As far as we know this is the first characterization of this type for graphs.

**Key words.** property testing, graph isomorphism, regularity lemma

**AMS subject classifications.** 68W25, 68Q25, 68R10

**DOI.** 10.1137/S009753970445603

## 1. Introduction.

Combinatorial property testing deals with the following task: For a fixed $\epsilon > 0$ and a fixed property $P$, distinguish using as few queries as possible (and with probability at least $\frac{2}{3}$) between the case that an input of length $m$ satisfies $P$, and the case that the input is $\epsilon$-far (with respect to an appropriate metric) from satisfying $P$. In our context the inputs are boolean functions, and the distance from $P$ is measured by the minimum number of bits that have to be modified in the input in order to make it satisfy $P$, divided by the input length $m$. Many cases of interest involve tests that have a number of queries that depends only on the approximation parameter $\epsilon$ and is independent of the input length. In this paper we define a measure of complexity for the properties we consider and use it to bound the number of queries required for the test.

We start with some historical background: The first time a question formulated in terms of property testing was considered was in the work of Blum, Luby, and Rubinfeld [3], and the general notion of property testing was first formally defined by Rubinfeld and Sudan [12], mainly for the context of the algebraic properties (such as linearity) of functions over finite fields and vector spaces. The first investigation in the combinatorial context was that of Goldreich, Goldwasser, and Ron [9], where the testing of combinatorial graph properties was first formalized; their framework will also be the one used here. In recent years the field of property testing has enjoyed rapid growth, as witnessed in the surveys [11] and [5].

In the context of boolean functions, it was proved in [6] that for a fixed $k$ and $\epsilon$ one can $\epsilon$-test a boolean function $f : \{0,1\}^n \to \{0,1\}$ for the property of depending on only $k$ of its variables, where the number of queries depends on only $\epsilon$ and $k$. As a corollary, it was then shown that if a function $h : \{0,1\}^n \to \{0,1\}$ is "simple," in the sense that it depends on only $k$ of its variables, then one can $\epsilon$-test an input function

[*]Received by the editors August 10, 2004; accepted for publication (in revised form) January 24, 2005; published electronically June 3, 2005. A preliminary version of this paper appeared in *Proceedings of the 36th Annual Symposium on the Theory of Computing (STOC)*, Chicago, IL, 2004.

http://www.siam.org/journals/sicomp/34-5/44560.html

[†]Faculty of Computer Science, Technion – Israel Institute of Technology, Technion City, Haifa 32000, Israel (eldar@cs.technion.ac.il). This research was supported in part by Israel Science Foundation grant 55/03 and by a grant from the Matilde Barnett Revocable Trust.

$f$, for the property of being identical to $h$ up to a permutation of its variables, using a number of queries that depends on only $k$ and $\epsilon$.

In [6] there were also examples of functions $h : \{0,1\}^n \to \{0,1\}$, depending on $k$ variables, for which the number of required queries is *at least* a function of $k$. However, not all functions that are far from depending on a few variables require many queries. The question of finding a complexity measure for which both upper and lower bounds hold for *every* function $h$ was posed in [6] and to our knowledge is still open.

Here we consider an analogous question for graphs: Given a graph $H$, how many queries are required to test an input graph $G$ for the property of being isomorphic to $H$? Here too, one would assume that the answer depends on some measure of the "complexity" of $H$.

First we need a criterion for simplicity, similar in spirit to that of depending on few variables in the case of boolean functions. Such a natural criterion is that the graph $H$ is the result of only a few basic operations (defined below), taken over a constant number of subsets of its vertices. Alternatively, one can formulate the criterion of $H$ having a partition of its vertex set into a constant number of subsets $W_1, \dots, W_k$, such that for every $1 \leq i \leq j \leq k$ the pair $W_i, W_j$ either contains no edge at all or contains all possible edges (the gap between the first and the second approach turns out to be not more than an exponent, which is irrelevant for the purpose here).

Our main result shows both an upper bound and a lower bound on the number of queries required for the test that a graph is isomorphic to $H$, given in terms of the complexity parameter of $H$. We thus obtain that a simple graph (or a graph that is close enough to simple) admits an easy isomorphism test, while a graph far from all simple graphs does not admit such a test. In contrast, the result from [6] about boolean functions provides only an upper bound.

The rest of the paper is organized as follows. Some preliminaries, the definition of our complexity measure for $H$, and the formal statement of the main result are all in section 2. The upper bound part of the main result is proved in section 3. Section 4 introduces Szemerédi's regularity lemma and other tools used in the proofs following it, and section 5 proves the lower bound part of our main result. All positive results here deal with 2-sided tests, which is not a coincidence as section 6 contains a simple proof of a nonconstant lower bound for 1-sided testing for isomorphism, even against a graph comprised of just one clique and an additional set of isolated vertices. Finally, section 7 contains some discussion and concluding comments.

We note here that the upper bound part of the main result is rather easy, using the results of [9]. The lower bound, however, uses Szemerédi's regularity lemma, known usually for its strength in proving *upper* bounds about graph property testing. This lemma, which enables us to find random-like structures in any graph, is used here to construct a "rerandomization" of $H$ that provides a graph $H'$ that is far from being isomorphic to $H$ and yet is not distinguishable from it by any testing algorithm using too few queries.

## 2. Preliminaries and statement of the main result.

The most central notion to property testing is that of the distance between inputs and properties. We deal with the "dense" model of graph property testing, first defined in [9], as per the following definition.

DEFINITION 1. *Given two (labeled) graphs $G$ and $G'$ on the same vertex set $V$, the* distance *between $G$ and $G'$ is the size of the symmetric difference between the edge sets of $G$ and $G'$, divided by $\binom{|V|}{2}$.*

*Given a graph $G$ and a graph property (a set of graphs that is closed under graph isomorphisms) $P$, the* distance *between $G$ and $P$ is the minimum distance between $G$ and any graph $G'$ on the same vertex set which satisfies $P$.*

Using this definition of the distance, we give a formal definition of a graph testing algorithm.

Definition 2. *An* $\epsilon$-testing algorithm with $q$ queries *for a property $P$ is a probabilistic algorithm that for any input graph $G$ makes up to $q$ queries (a query consisting of finding whether two vertices $u, v$ of $G$ form an edge of $G$ or not) and satisfies the following.*

1. *If $G$ satisfies $P$, then the algorithm accepts $G$ with probability at least $\frac{2}{3}$.*
2. *If $G$ is $\epsilon$-far from $P$, that is, the distance of $G$ from $P$ is more than $\epsilon$, then the algorithm rejects $G$ with probability at least $\frac{2}{3}$.*

The measure of the complexity of a graph $H$ is formalized in the following. We also formulate an approximate notion, of only being close to a simple graph. We need this technicality because such graphs also admit an efficient isomorphism test, as one can just test the input graph for isomorphism with the simple graph close to $H$, instead of testing for isomorphism with $H$ itself.

Definition 3. *The* algebra number *of a graph $H$, denoted by* $\mathrm{Algnum}(H)$*, is the minimal number $k$ for which there exist cliques $H_1, \dots, H_k$ over subsets of the vertex set of $H$, such that $H$ is in the boolean algebra generated by $H_1, \dots, H_k$ (that is, the edge set of $H$ can be generated from the edge sets of $H_1, \dots, H_k$ by the appropriate set operations).*

*The* $\epsilon$-approximate algebra number*, denoted by* $\mathrm{Algnum}_\epsilon(H)$*, is the minimal $k$ such that $H$ is $\epsilon$-close (i.e., has distance at most $\epsilon$) to some graph whose algebra number is $k$.*

For stating the main result, it is convenient to define the number of queries required for $\epsilon$-testing a graph for the property of being isomorphic to $H$ as a parameter of $H$.

Definition 4. *The* $\epsilon$-testing number *of a graph $H$, denoted by* $\mathrm{Testnum}_\epsilon(H)$*, is the minimum $q$ for which there exists an $\epsilon$-testing algorithm with $q$ queries for the property of being isomorphic to $H$.*

We are now ready for the formal statement of the main result.

Theorem 2.1. *For every $\epsilon$ there exists a pair of functions $L_\epsilon(t)$ and $U_\epsilon(t)$, with $\lim_{t\to\infty} L_\epsilon(t) = \infty$, such that for every graph $H$ we have* $L_\epsilon(\mathrm{Algnum}_{3\epsilon}(H)) \leq \mathrm{Testnum}_\epsilon(H) \leq U_\epsilon(\mathrm{Algnum}_{\epsilon/3}(H))$.

As a final note, it is not a coincidence that our results provide 2-sided algorithms, that is, algorithms that may err with some small probability on both sides. Section 6 contains a proof that 1-sided algorithms (algorithms that accept a graph that is isomorphic to $H$ with probability 1) require an unbounded number of queries even for a graph $H$ for which $\mathrm{Algnum}(H) = 1$.

## 3. Isomorphism testing against a simple graph.

To prove that it is easy to test an input graph for isomorphism against a simple graph that is given in advance, we use the result of [9] about the efficiency of testing for properties defined by the existence of a partition with prescribed densities, as stated in the following.

Definition 5. *For two disjoint sets $U, W$ of vertices of a graph $G$ we define $d(U, W)$ as the number of edges between $U$ and $W$, divided by $|U||W|$. In addition we define $d(U, U)$ for every $U$ as the number of edges of $G$ internal to $U$, divided by $\binom{|U|}{2}$.*

*Suppose we are given parameters $m$, $(\alpha_i)_{1\le i\le m}$, $(\alpha'_i)_{1\le i\le m}$, $(\beta_{i,j})_{1\le i\le j\le m}$, and $(\beta'_{i,j})_{1\le i\le j\le m}$ satisfying $0 \le \alpha_i < \alpha'_i \le 1$ and $0 \le \beta_{i,j} < \beta'_{i,j} \le 1$. We say that a graph $G$ satisfies the* partition property *with respect to these parameters if there exists a partition $V_1,\dots,V_m$ of the vertex set of $G$ such that $\alpha_i \le \frac{1}{n}|V_i| \le \alpha'_i$ for every $1 \le i \le m$, and $\beta_{i,j} \le d(V_i,V_j) \le \beta'_{i,j}$ for every $1 \le i \le j \le m$.*

Lemma 3.1 (partition testing [9]). *Given parameters $m$, $(\alpha_i)_{1\le i\le m}$, $(\alpha'_i)_{1\le i\le m}$, $(\beta_{i,j})_{1\le i\le j\le m}$, and $(\beta'_{i,j})_{1\le i\le j\le m}$ satisfying $0 \le \alpha_i < \alpha'_i \le 1$ and $0 \le \beta_{i,j} < \beta'_{i,j} \le 1$, the partition property with respect to these parameters is testable with $\epsilon^{-\mathrm{poly}(m)}$ queries, a number that depends on only $m$ and the distance parameter $\epsilon$.*

We first prove as a warm-up that if $H$ has a small algebra number (and not just a small approximated algebra number), then there is a test for being an isomorphism of $H$ that uses few queries. The following lemma shows in essence that a graph with a small algebra number can be approximated by a partition property (in fact if we were allowed to choose $\alpha_i = \alpha'_i$ and $\beta_{i,j} = \beta'_{i,j}$, then there would be a partition property *identical* to the property of being an isomorphism of $H$).

Lemma 3.2. *If* $\mathrm{Algnum}(H) = k$, *then for every $\epsilon$, there exist for some $m \le 2^k$ parameters for the partition property (as it appears in Lemma 3.1) satisfying the following.*

*$H$ satisfies the partition property with respect to the parameters, and, moreover, every graph $G$ that satisfies it is $\frac{1}{2}\epsilon$-close to being isomorphic to $H$.*

*Proof.* We look at the construction of $H$ from the cliques $H_1,\dots,H_k$, and let $W_1,\dots,W_m$ be the atoms of the boolean algebra generated from the vertex sets of $H$ and $H_1,\dots,H_k$ (in other words, $W_1,\dots,W_m$ are the smallest nonempty vertex sets that can be constructed from the vertices of $H$ and $H_1,\dots,H_k$ by set operations). Note that $m \le 2^k$. We now note that for this partition, for every $1 \le i \le j \le m$ the graph $H$ either has all possible edges between $W_i$ and $W_j$, or it has no such edge. From this it is not hard to see that the following parameters will satisfy the assertions of the lemma: $\alpha_i = \max\{0, \frac{1}{n}|W_i| - \frac{1}{8m}\epsilon\}$, $\alpha'_i = \min\{1, \frac{1}{n}|W_i| + \frac{1}{8m}\epsilon\}$, $\beta_{i,j} = \max\{0, d(W_i,W_j) - \frac{1}{4}\epsilon\}$, and $\beta'_{i,j} = \min\{1, d(W_i,W_j) + \frac{1}{4}\epsilon\}$ (note that here there are only two possible values for $\beta_{i,j}$ and $\beta'_{i,j}$, as $d(W_i,W_j)$ is either 1 or 0). □

From Lemma 3.2 it is not hard to construct an $\epsilon$-test for the property of being an isomorphism of an $H$ with a small algebra number—we just apply a $\frac{1}{2}\epsilon$-test as guaranteed by Lemma 3.1 for the property given by Lemma 3.2. Such a test uses for a fixed $\epsilon$ a number of queries that is exponential in some power of $m \le 2^{\mathrm{Algnum}(H)}$.

The following is a strengthening of Lemma 3.2, in that it holds also for graphs with a small *approximated* algebra number.

Lemma 3.3. *If* $\mathrm{Algnum}_{\epsilon/3}(H) = k$, *then there exist partition parameters as in the formulation of Lemma 3.1, where $m \le 2^k$, such that the partition property with respect to these parameters satisfies the following.*

*$H$ satisfies the partition property, and, moreover, every graph that satisfies it is $\frac{8}{9}\epsilon$-close to being isomorphic to $H$.*

*Proof.* We again construct $W_1,\dots,W_m$ as the smallest nonempty vertex sets that can be generated from the vertex sets of the appropriate cliques, but instead of doing this for $H$ we do it for a graph $H'$ that is $\frac{1}{3}\epsilon$-close to $H$ and whose algebra number is $k$. We then set $\alpha_i = \max\{0, \frac{1}{n}|W_i| - \frac{1}{18m}\epsilon\}$, $\alpha'_i = \min\{1, \frac{1}{n}|W_i| + \frac{1}{18m}\epsilon\}$, $\beta_{i,j} = \max\{0, d(W_i,W_j) - \frac{1}{9}\epsilon\}$, and $\beta'_{i,j} = \min\{1, d(W_i,W_j) + \frac{1}{9}\epsilon\}$, where $d(W_i,W_j)$ denotes the density of the respective pair in the graph $H$ and not in $H'$ (this is important).

Again it is easy to see that $H$ satisfies the partition property. On the other hand, any graph that satisfies the partition property is $\frac{2}{9}\epsilon$-close to some graph that has a partition into $W_1, \ldots, W_m$ with exactly the same set sizes as $H$ and exactly the same pair densities as $H$.

A graph $\tilde{H}$ having the same densities as $H$ is $\frac{2}{3}\epsilon$-close to it because $H$ is $\frac{1}{3}\epsilon$ close to $H'$: For every $W_i, W_j$, the number of vertex pairs between $W_i$ and $W_j$ for which $H$ and $\tilde{H}$ differ is clearly bounded by $2\min\{d(W_i, W_j), 1 - d(W_i, W_j)\}$ times the total number of vertex pairs ($|W_i||W_j|$ if $i < j$ and $\binom{|W_i|}{2}$ if $i = j$), and by the information on $H$ the sum of these differences when taken over all $1 \le i \le j \le m$ is bounded by $\frac{2}{3}\epsilon\binom{n}{2}$. From this we conclude that a graph that satisfies the partition property is $\frac{8}{9}\epsilon$-close to being isomorphic to $H$. $\square$

PROPOSITION 3.4 (part 1 of Theorem 2.1). *For every fixed $\epsilon$ there exists a function $U_\epsilon(t)$ such that* $\text{Testnum}_\epsilon(H) \le U_\epsilon(\text{Algnum}_{\epsilon/3}(H))$ *for every graph $H$.*

*Proof.* If $\text{Algnum}_{\epsilon/3}(H) \le k$, then we construct a test whose number of queries depends on only $\epsilon$ and $k$ as follows. We take the partition property provided for $H$ by Lemma 3.3 and $\frac{1}{9}\epsilon$-test for it using the testing algorithm provided by Lemma 3.1. It is clear why this algorithm accepts (with probability at least $\frac{2}{3}$) a graph that is isomorphic to $H$ (and hence satisfies the partition property); on the other hand, if the input graph $G$ is $\epsilon$-far from being isomorphic to $H$, then by Lemma 3.3 and the triangle inequality it is at least $\frac{1}{9}\epsilon$-far from the partition property, and so it is rejected by the algorithm. $\square$

We close this section with some remarks about the running time of the testing algorithm. In general, the partition testing algorithm provided in [9] consists of choosing a uniformly random set $U$ of vertices of $G$, with a constant number of vertices (this number is a polynomial in $\epsilon$ whose coefficients and degree depend on $m$), and then checking all possible partitions of the subgraph induced by $U$ for certain properties. The running time of such a test is thus exponential in the number of queries.

Given a bound on $\text{Algnum}(H)$ (and not just on the approximated algebra number), it seems that by the methods of [9] one can devise a test in which only partitions of the induced subgraph into $m$ sets, for which all set pairs are either full (of density 1) or edgeless (of density 0), need be considered. These partitions can be calculated in only a polynomial time in the number of queries, by sorting the vertices according to their sets of neighbors, yielding a corresponding reduction in the running time of the test.

## 4. Further preliminaries and Szemerédi's regularity lemma.

The most common use of Szemerédi's regularity lemma in graph property testing is in proving upper bounds such as, e.g., those in [1] and [4]. Here it will be used for proving a lower bound—the existence of regular pairs (as defined below) will be used in constructing a graph that is far from the original $H$, but in a way that is not detectable by a testing algorithm. First we define the regularity of pairs.

DEFINITION 6. *For two nonempty disjoint vertex sets $A$ and $B$ of a graph $H$, we say that the pair $A, B$ is* $\gamma$-regular *if, for any two subsets $A'$ of $A$ and $B'$ of $B$ satisfying $|A'| \ge \gamma|A|$ and $|B'| \ge \gamma|B|$, the edge densities satisfy $|d(A', B') - d(A, B)| < \gamma$.*

A first observation about regular pairs is the following well-known fact that regularity is somewhat preserved when considering large enough subsets of the original pair.

*Observation* 4.1. If $A, B$ is a $\gamma$-regular pair with density $\eta$ and $A' \subset A$ and $B' \subset B$ satisfy $|A'| \ge \delta|A|$ and $|B'| \ge \delta|B|$ for some $\delta \ge \gamma$, then $A', B'$ is a $\max\{2, \delta^{-1}\}\gamma$-regular pair, with density at least $\eta - \gamma$ and at most $\eta + \gamma$. $\square$

The following lemma shows how much regular pairs have in common with random bipartite graphs. Its proof is not new (similar lemmas have been used extensively in the literature), but it is given here for completeness. To clarify the presentation of the mathematical formulas in the following, functions that are defined in a statement of a lemma are indexed with the lemma's number.

Lemma 4.2. *For every fixed $t$ and $\epsilon > 0$ there exists $\gamma = \gamma_{4.2}(\epsilon, t)$ with the following property.*

*Suppose that $I$ is a graph with vertices $v_1, \dots, v_t$ and that $V_1, \dots, V_t$ is a $t$-tuple of disjoint vertex sets of $H$ such that for every $1 \le i < j \le t$ the pair $V_i, V_j$ is $\gamma$-regular. Define $\eta_{i,j}$ to be $d(V_i, V_j)$ if $v_i v_j$ is an edge of $I$ and $1 - d(V_i, V_j)$ if $v_i v_j$ is not an edge of $I$. Then, the number of $t$-tuples $w_1 \in V_1, \dots, w_t \in V_t$ that span induced copies of $I$, where each $w_i$ plays the role of $v_i$, is between $(\prod_{1 \le i < j \le t} \eta_{i,j} - \epsilon) \prod_{l=1}^{t} |V_l|$ and $(\prod_{1 \le i < j \le t} \eta_{i,j} + \epsilon) \prod_{l=1}^{t} |V_l|$.*

*Proof.* We will prove here the existence of $\gamma' = \gamma'(\epsilon, t)$ such that the number of copies of $I$ as per the assertion of the lemma is at least $(\prod_{1 \le i < j \le t} \eta_{i,j} - \epsilon) \prod_{l=1}^{t} |V_l|$. We can then set $\gamma = \gamma'(2^{-\binom{t}{2}}\epsilon, t)$, because then the full assertion of the lemma (both upper and lower bounds) follows from the correctness of the above for every one of the $2^{\binom{t}{2}}$ possible graphs $I$.

In proving the existence of $\gamma'$ we assume without loss of generality that $I$ is the clique with $t$ vertices. In this case we may also assume that $\eta_{i,j} = d(V_i, V_j) > \epsilon$ for every $1 \le i < j \le t$, as otherwise the above assertion is trivial. The proof is by induction on $t$, where the case $t = 2$ is trivial.

We set $\gamma' = \min\{\epsilon/4\binom{t-1}{2}, \epsilon/4(t-1), \frac{3}{4}\epsilon\gamma'(\frac{1}{4}\epsilon, t-1)\}$. We look at the vertices of $V_t$. Because of the regularity property, for every $1 \le i < t$ there are at least $(1 - \epsilon/4(t-1))|V_t|$ vertices of $V_t$ that have at least $(\eta_{i,t} - \epsilon/4(t-1))|V_i|$ neighbors in $V_i$, as otherwise the set $V_t'$ of vertices that do not satisfy this with the set $V_i$ will contradict the regularity of the pair $V_i, V_t$. Thus there are at least $(1 - \frac{1}{4}\epsilon)|V_t|$ vertices of $V_t$ for which the degree condition is true for *every* $1 \le i < t$.

For every vertex $v$ of $V_t$ satisfying the above, let $V_i'$ be the set of neighbors of $v$ in $V_i$ for $1 \le i \le t-1$, and accordingly set $\eta_{i,j}' = d(V_i', V_j')$ for every $1 \le i < j \le t-1$. We note that $|V_i'| \ge (\eta_{i,t} - \epsilon/4(t-1))|V_i|$ (and in particular $|V_i'| \ge \frac{3}{4}\epsilon|V_i|$) for every $i$, that $\eta_{i,j}' \ge \eta_{i,j} - \epsilon/4\binom{t-1}{2}$ for every $i < j$, and that every pair $V_i', V_j'$ is in particular $\gamma'(\frac{1}{4}\epsilon, t-1)$-regular by Observation 4.1. We now use the induction hypothesis to obtain that $V_1', \dots, V_{t-1}'$ admit sufficiently many cliques with $t-1$ vertices, one from every $V_i'$, where each such clique makes a copy of $I$ with $v \in V_t$ added. Specifically, the number of such cliques is at least

$$\left(\prod_{1 \le i < j \le t-1} \eta_{i,j}' - \frac{1}{4}\epsilon\right) \prod_{l=1}^{t-1} |V_l'| \ge \left(\prod_{1 \le i < j \le t-1} \eta_{i,j} - \frac{2}{4}\epsilon\right) \prod_{l=1}^{t-1} |V_l'|$$
$$\ge \left(\prod_{1 \le i < j \le t} \eta_{i,j} - \frac{3}{4}\epsilon\right) \prod_{l=1}^{t-1} |V_l|.$$

Multiplying this by the number of vertices $v \in V_t$ for which the analysis holds, we obtain the required number of copies of $I$. □

In our lower bound proof we construct a graph that is hard to distinguish from $H$ by a testing algorithm. We construct it by replacing some of the regular pairs of $H$ with actual random subgraphs. But to find these regular pairs, we need Szemerédi's regularity lemma.

DEFINITION 7. *An* equipartition *of a graph $H$ is a partition of its vertex set into sets whose sizes differ from each other by no more than* 1. *A $\gamma$*-regular partition *is an equipartition of $H$ into $k$ sets such that all $\binom{k}{2}$ set pairs but at most $\gamma\binom{k}{2}$ of them are $\gamma$-regular.*

LEMMA 4.3 (Szemerédi's regularity lemma [13]). *For every $m$ and $\gamma > 0$ there exists $T = T_{4.3}(m, \gamma)$ with the following property.*

*If $H$ is a graph with $n \geq T$ vertices, and $\mathcal{A}$ is an equipartition of the vertex set of $H$ into $m$ sets, then there exists an equipartition $\mathcal{B}$ that is a refinement of $\mathcal{A}$ into $k$ sets, where $m \leq k \leq T$, and such that $\mathcal{B}$ is a $\gamma$-regular partition.*

For the proof of the lower bound we will need an equipartition of the vertices of $H$ so that many of its pairs, apart from being regular, also have densities that are not too close to 0 or 1. A high approximate algebra number guarantees this.

LEMMA 4.4. *For every $\epsilon > 0$, $\gamma > 0$, and $m$ there exists $T = T_{4.4}(\epsilon, \gamma, m)$ such that if $H$ is a graph with $\mathrm{Algnum}_{3\epsilon}(H) > T$, then there exists a $\gamma$-regular partition $\mathcal{B} = \{V_1, \ldots, V_k\}$ of $H$ with $m \leq k \leq T$, for which*

$$\binom{k}{2}^{-1} \sum_{1 \leq i < j \leq k} \min\{d(V_i, V_j), 1 - d(V_i, V_j)\} > 2\epsilon.$$

*Proof.* We set $T = \binom{T_{4.3}(\max\{m, 2/\epsilon\}, \gamma) + 1}{2}$. We first use Lemma 4.3 to find the appropriate $\gamma$-regular partition $\mathcal{B}$ with $k$ sets, where $k \leq T_{4.3}(\max\{m, 2/\epsilon\}, \gamma)$ (we let $\mathcal{A}$ be an arbitrary equipartition with $\max\{m, 2/\epsilon\}$ sets), and then prove that $\mathrm{Algnum}_{3\epsilon}(H) \leq \binom{k+1}{2}$ unless $\mathcal{B}$ satisfies the assertion of the lemma.

We let $H'$ be the graph obtained from $H$ by the following modifications: For every $i$ we add all possible edges inside $V_i$, and in addition for every pair $V_i, V_j$, if $d(V_i, V_j) > \frac{1}{2}$, then we add all possible edges between $V_i$ and $V_j$, and if $d(V_i, V_j) \leq \frac{1}{2}$, then we remove all edges between these sets.

The distance between $H'$ and $H$ is bounded by

$$\epsilon + \binom{k}{2}^{-1} \sum_{1 \leq i < j \leq k} \min\{d(V_i, V_j), 1 - d(V_i, V_j)\}.$$

On the other hand, the algebra number of $H'$ is no more than $\binom{k+1}{2}$, because its edge set is the union of the edge sets of the following cliques: the clique on $V_i$ for every $1 \leq i \leq k$ and the clique on $V_i \cup V_j$ for every $1 \leq i < j \leq k$ for which $d(V_i, V_j) > \frac{1}{2}$. This means that if the assertion of the lemma does not hold, then $\mathrm{Algnum}_{3\epsilon}(H) \leq \binom{k+1}{2}$, a contradiction. $\square$

To prove lower bounds it is best if we can restrict ourselves to relatively simple algorithms. In the context of graph properties, this can be done with the following.

LEMMA 4.5 (from [10]). *If there exists an $\epsilon$-testing algorithm for a graph property that makes $q$ queries, then there exists an $\epsilon$-testing algorithm that makes its queries by uniformly and randomly choosing a set of $2q$ vertices and querying all their pairs. In particular, it is a nonadaptive $\epsilon$-test making $\binom{2q}{2}$ queries.*

## 5. Isomorphism testing against a complex graph.

Given a graph $H$, we would like to construct a graph $H'$ that is far from being isomorphic to $H$, but in a way that cannot be detected by a testing algorithm of the type depicted in Lemma 4.5. The way to do this is to replace regular pairs in $H$ with alternative regular pairs. We would like to construct them at random; for such considerations the following well-known large deviation inequality comes in handy.

LEMMA 5.1 (see, e.g., [2, Appendix A]). *Suppose that we are given $m$ independent random variables $X_1, \dots, X_m$, each of which receiving a value of 1 with probability $p_i$ and a value of 0 with probability $1 - p_i$. The probability that $|\frac{1}{m}\sum_{i=1}^m X_i - \frac{1}{m}\sum_{i=1}^m p_i| > \delta$ is bounded by $2e^{-2\delta^2 m}$.*

The following lemma now allows us to construct regular pairs at random.

LEMMA 5.2. *For every $\gamma > 0$ and $\epsilon > 0$ there exists $\alpha = \alpha_{5.2}(\gamma, \epsilon) > 0$ and $N = N_{5.2}(\gamma, \epsilon)$ with the following property: If $I$ is a random bipartite graph with color classes $A$ and $B$ where $|A| \ge N$ and $|B| \ge N$, in which every pair of vertices is taken to be an edge independently with probability $\eta$, then with probability at least $1 - 2^{-\alpha|A||B|}$ the pair $A, B$ will be a $\gamma$-regular pair (with respect to $I$) whose density is between $\eta - \epsilon$ and $\eta + \epsilon$.*

*Proof.* For proving the required regularity and density properties, it is not hard to see that it is enough to prove that with the asserted probability, every $A' \subset A$ with $|A'| = \gamma|A|$ and $B' \subset B$ with $|B'| = \gamma|B|$ satisfy $|d(A', B') - \eta| \le \min\{\frac{1}{2}\gamma, \epsilon\}$. For every fixed $A'$ and $B'$, using Lemma 5.1 (where the variables $X_1, \dots, X_m$ are defined as the indicator variables for the edges between $|A'|$ and $|B'|$) implies that the probability that the above is not satisfied is at most $2e^{-2(\min\{\gamma/2,\epsilon\})^2|A'||B'|} \le 2^{-\beta|A'||B'|} = 2^{-\beta'|A||B|}$ for the appropriate $\beta$ and $\beta' = \gamma^2\beta$. To bound the probability that the above is not satisfied for any of the possible pairs $|A'|, |B'|$, whose number is no more than $2^{|A|+|B|}$, we set $\alpha = \frac{1}{2}\beta'$ and $N = 4/\beta'$ (for which $|A| + |B| \le \frac{1}{2}\beta'|A||B|$ holds). $\square$

We also need the following rather trivial observation about distance of random graphs.

*Observation* 5.3. For every $\epsilon$ there exists $\alpha = \alpha_{5.3}(\epsilon) > 0$ such that if $J$ is any fixed bipartite graph with vertex classes $A$ and $B$, and $I$ is a random bipartite graph with the same vertex classes, where every pair of vertices is taken to be an edge independently with probability $\eta$, then with probability at least $1 - 2^{-\alpha|A||B|}$, the number of pairs that belong to the symmetric difference between the edges of $I$ and $J$ is at least $(\min\{\eta, 1-\eta\} - \epsilon)|A||B|$.

*Proof sketch.* Assume without loss of generality that $\eta \le \frac{1}{2}$, and define the random variables $X_1, \dots, X_m$ as the indicator variables, each for the event that a specific vertex pair belongs to the symmetric difference between the edges of $I$ and those of the original graph $J$. One can easily see that regardless of the original graph, $\frac{1}{m}\sum_{i=1}^m p_i \ge \eta$, and then use Lemma 5.1. $\square$

We now formalize which features of an input graph a testing algorithm of the type depicted in Lemma 4.5 can detect.

DEFINITION 8. *The* $t$-statistic *of a graph $H$ is the distribution over labeled graphs with $t$ vertices that results from choosing uniformly an ordered set (with no repetitions) of $t$ vertices of $H$, and looking at the appropriate induced subgraph.*

*The* variation distance *between the $t$-statistics of a graph $H$ and a graph $H'$ is defined as the usual variation distance between the corresponding distributions. That is, if we denote by $\mu$ the $t$-statistic of $H$ and by $\mu'$ the $t$-statistic of $H'$, then the variation distance is equal to $\frac{1}{2}\sum_K |\mathrm{Pr}_\mu[K] - \mathrm{Pr}_{\mu'}[K]|$, where the index $K$ runs over possible labeled graphs with a fixed set of $t$ vertices.*

The following lemma is then immediate from Lemma 4.5.

LEMMA 5.4. *If $H$ and $H'$ have $2q$-statistics which differ by less than $\frac{1}{3}$ in the variation distance and $H'$ is $\epsilon$-far from being isomorphic to $H$, then* $\mathrm{Testnum}_\epsilon(H) > q$.

*Proof.* If there exists an $\epsilon$-test for the property of being isomorphic to $H$ that makes at most $q$ queries, then by Lemma 4.5 there exists such a test that chooses a uniformly random set of $2q$ vertices and bases its decision solely on the subgraph induced on this set. However, if such an algorithm accepts $H$ with probability at least $\frac{2}{3}$, then the variation distance between the $2q$-statistics of $H$ and $H'$ implies that the algorithm accepts $H'$ with probability more than $\frac{1}{3}$, a contradiction. ∎

The main lemma is the following. It shows how, given a regular partition of a graph in which many pairs have densities that are not close to 0 or 1, one can construct a graph $H'$ that is the one we need for a lower bound on the testing number.

LEMMA 5.5. *For every $t$ and $\epsilon$ there exist $m = m_{5.5}(t, \epsilon)$ and $\gamma = \gamma_{5.5}(t, \epsilon)$, such that for every $T \geq m$ there exists $N = N_{5.5}(t, \epsilon, T)$ with the following property. Suppose that for a graph $H$ with $n > N$ vertices there exists a $\gamma$-regular partition $V_1, \dots, V_k$ with $m \leq k \leq T$ that also satisfies $\binom{k}{2}^{-1} \sum_{1 \leq i < j \leq k} \min\{d(V_i, V_j), 1 - d(V_i, V_j)\} > 2\epsilon$. Then there exists a graph $H'$ that is $\epsilon$-far from being isomorphic to $H$, and such that the $t$-statistics of the two graphs have variation distance less than $\frac{1}{3}$ between them.*

*Proof.* We choose

$$\gamma = \min\left\{2^{-\binom{t}{2}}/15\binom{t}{2}, \gamma_{4.2}\left(2^{-\binom{t}{2}}/15\binom{t}{2}, t\right), \frac{1}{3}\epsilon\right\}$$

and

$$m = \max\left\{15\binom{t}{2}, 3/\epsilon\right\}.$$

$H'$ is constructed from $H$ as follows: For every $\gamma$-regular pair $V_i, V_j$ in the equipartition of $H$, we let the edges of $H'$ between $V_i$ and $V_j$ be randomly and independently taken with probability $\eta_{i,j} = d(V_i, V_j)$ each. For every $i, j$ such that $i = j$ or $V_i, V_j$ is not a regular pair, we just let the corresponding edges of $H'$ be identical to those of $H$ (this is an arbitrary choice, and any other choice for such a pair will also do here).

To analyze the $t$-statistics we consider a uniformly random ordered set of vertices (with no repetitions), $v_1, \dots, v_t$. For $1 \leq j \leq t$ we define $i_j$ to be such that $v_j \in V_{i_j}$; with probability at least $\frac{14}{15}$ the indexes $i_1, \dots, i_t$ will be all different, because $m \geq 15\binom{t}{2}$. Given this event, with probability at least $\frac{14}{15}$ (for the choice of $v_1, \dots, v_t$) all the pairs in $V_{i_1}, \dots, V_{i_t}$ are $\gamma$-regular with respect to $H$, because $\gamma \leq 1/30\binom{t}{2}$.

We assume that $N$ is, in particular, chosen to be large enough so that with probability at least $\frac{3}{4}$ over the choice of $H'$, the following holds: Every pair $V_i, V_j$ that was $\gamma$-regular with respect to $H$ will also be $\gamma$-regular with respect to $H'$, and its density will be between $\eta_{i,j} - 2^{-\binom{t}{2}}/15\binom{t}{2}$ and $\eta_{i,j} + 2^{-\binom{t}{2}}/15\binom{t}{2}$, on account of Lemma 5.2 (we choose $N$ so that the probability for every such pair to be otherwise is bounded by $1/4\binom{T}{2}$).

The above choices guarantee that with probability at least $\frac{3}{4}$ over the choice of $H'$, the graphs $H$ and $H'$ have $t$-statistics that differ by less than $\frac{1}{3}$. This is because when we condition the choice of $v_1, \dots, v_t$ on a particular outcome of $i_1, \dots, i_t$ that are all different and for which all pairs are $\gamma$-regular (in both graphs), Lemma 4.2 guarantees that the two conditioned distributions on the induced subgraphs differ from each other by no more than 3/15, because for every fixed labeled graph with $t$ vertices the respective probabilities for its occurrence differ by no more than $3 \cdot 2^{-\binom{t}{2}}/15$. The bound on the difference is because Lemma 4.2 is invoked once for $H$ and once for $H'$ (accounting for a possible $2^{-\binom{t}{2}}/15$ difference from the corresponding product of pair densities in each invocation), while there may be up to an additional $2^{-\binom{t}{2}}/15$ difference between the products of the pair densities for $H$ and $H'$.

To make sure that $H$ and $H'$ are also $\epsilon$-far from being isomorphic, in addition to the choice of the parameters above, we assume that $N$ is chosen to be large enough so that with probability at least $\frac{3}{4}$ over the choice of $H'$, the following occurs: For every pair $V_i, V_j$ of $H'$ that corresponds to a $\gamma$-regular pair of $H$, when its edges are compared to the edges of $H$ between any two disjoint labeled vertex sets $U_1, U_2$ of the same sizes as $V_i, V_j$, the number of vertex pairs in the symmetric difference will be at least $(\min\{\eta_{i,j}, 1-\eta_{i,j}\} - \frac{1}{3}\epsilon)|V_i||V_j|$. A large enough $N$ will satisfy this, because the number of all possibilities for choosing $U_1$ and $U_2$ and labeling them for correspondence with $V_i$ and $V_j$, respectively, is clearly bounded by $n!$, while the bound on the probability for a lesser difference between the edges of $H'$ over $V_i, V_j$ and the edges of $H$ over $U_1, U_2$ is $2^{-\Theta(n^2)}$ by Observation 5.3, which decreases more rapidly than $1/n!$.

When this occurs, there is at least a $\binom{k}{2}^{-1}\big(\sum_{1\le i<j\le k}\min\{\eta_{i,j}, 1-\eta_{i,j}\}\big) - \epsilon \ge \epsilon$ distance between $H$ and $H'$, so with probability at least $\frac{1}{2}$ the graph $H'$ has all the required properties. ∎

Proposition 5.6 (part 2 of Theorem 2.1). *For every fixed $\epsilon$ there exists a function $L_\epsilon(t)$, with $\lim_{t\to\infty} L_\epsilon(t) = \infty$, such that $L_\epsilon(\mathrm{Algnum}_{3\epsilon}(H)) \le \mathrm{Testnum}_\epsilon(H)$ for every graph $H$.*

*Proof.* We construct the inverse function of $L_\epsilon(t)$, that is, we prove that for every $q$ there exists $A = A_\epsilon(q)$ such that if $\mathrm{Algnum}_{3\epsilon}(H) > A$, then $\mathrm{Testnum}_\epsilon(H) > q$. We will first prove the above only for graphs with $n > N$ vertices for some $N = N_\epsilon(q)$. From this we can conclude the proof, as we can use $A' = \max\{A, \binom{N}{2}\}$ instead of $A$ and then remove the condition that $n > N$, because clearly $\mathrm{Algnum}(G) \le \binom{n}{2}$ for any graph $G$ with $n$ vertices. We now choose $A = T_{4.4}(\epsilon, \gamma_{5.5}(2q,\epsilon), m_{5.5}(2q,\epsilon))$ and $N = N_{5.5}(2q, \epsilon, A)$.

If $H$ has $n > N$ vertices and satisfies $\mathrm{Algnum}_{3\epsilon}(H) > A$, we construct an $H'$ that is $\epsilon$-far from being isomorphic to $H$, and such that the $2q$-statistics of $H$ and $H'$ differ by less than $\frac{1}{3}$, which by Lemma 5.4 implies that $\mathrm{Testnum}_\epsilon(H) > q$. For this we use Lemma 4.4 to find a partition satisfying the conditions of Lemma 5.5, through which we construct $H'$. ∎

## 6. The difficulty of 1-sided testing for isomorphism against H.

The test constructed in section 3 for graphs with a small approximate algebra number has a 2-sided error, and the lower bounds for graphs with a large approximate algebra number hold for the general 2-sided error algorithms as well. If one insists on the more restricted framework of a 1-sided testing algorithm, where the testing algorithm must accept a graph that is isomorphic to $H$ with probability 1, then the situation is much worse, as a bounded $\mathrm{Algnum}(H)$ does not guarantee any bound on the testing number of $H$.

*Observation* 6.1. Let $H$ be a graph with $n$ vertices, constructed by taking a clique with $\frac{1}{2}n$ vertices and adding to it $\frac{1}{2}n$ additional isolated vertices (note that in particular $\mathrm{Algnum}(H) = 1$). A 1-sided $\frac{1}{4}$-test for being isomorphic to $H$ requires at least $\frac{1}{4}n$ queries.

*Proof.* If $G$ is a random permutation of $H$, then any test making fewer than $\frac{1}{4}n$ queries will, with some positive (albeit small) probability, fail to find any edge of $G$. This is because the $\frac{1}{4}n$ queries of the algorithm cannot involve more than $\frac{1}{2}n$ vertices, and with some (small but positive) probability none of these vertices will belong to the clique. If the algorithm has 1-sided error, then it must accept $G$ with probability 1 even when encountering this small probability case. But this means that the algorithm will also accept the null (edgeless) graph with $n$ vertices with probability 1, which is a contradiction. $\square$

In fact, *any* graph $H$ with $n$ vertices that admits a 1-sided $\epsilon$-test for isomorphism making $o(\log n)$ queries is either $\epsilon$-close to being a clique or $\epsilon$-close to being edgeless for $n$ large enough. To prove this, one can use Ramsey's theorem to find in $H$ either a clique with $\Omega(\log n)$ vertices or an edgeless set of this size, and then proceed similarly to the proof above to show that the 1-sided $\epsilon$-testing algorithm must also accept either the clique with $n$ vertices or the edgeless graph with $n$ vertices with probability 1.

## 7. Concluding comments.

**The question of a combinatorial characterization of all testable graph properties.** The question of a logical characterization of the testable graph properties seems very hard. Just as this work provides a combinatorial characterization for a graph admitting an easy isomorphism test, one could hope for a combinatorial characterization of whole properties for admitting an easy test, which motivated a joint work with Newman [8]. It is hinted there that a testable graph property may be characterized as being approximable by properties concerning regular graph partitions.

The resulting "characterization" is, however, convoluted and not really a characterization in the strictest sense of the word. However, for the purpose of [8] it was used to prove that for every testable graph property, one can actually approximate the distance of any graph from satisfying it using a number of queries that depends only on the (additive) tolerance.

**Worst cases and unknown graphs.** After showing that the testing number of a graph depends on a measure of its complexity, there is still the question of the worst case. In other words, there is the question of estimating the maximum for $\text{Testnum}_\epsilon(H)$ where $\epsilon$ is fixed and $H$ ranges over all graphs with $n$ vertices.

If we disregard running time, then it is not hard to construct an $\epsilon$-testing algorithm that uses $\tilde{O}(n)$ queries for any graph $H$ that is given in advance, where the tilde notation here and in the following is used to hide polylogarithmic factors in $n$. This holds because, for an input graph $G$ and a graph $H$ that is given in advance, and for a particular labeling of the vertices of $G$, it takes $\tilde{O}(n)$ queries to detect with probability at least $1 - \frac{1}{3n!}$ whether $G$ differs from $H$ in at least $\epsilon\binom{n}{2}$ labeled pairs. By using a union bound over all $n!$ possible labelings of the vertices of $G$, one set of $\tilde{O}(n)$ queries is sufficient to completely test $G$ for having any labeling that can serve as an isomorphism function with $H$. A joint work with A. Matzliah [7] suggests that the true bound for 2-sided error algorithms is in fact a smaller power of $n$.

Another problem is deciding whether two graphs $G$ and $H$ that are both given as input (with only the number of vertices $n$ being given in advance) are isomorphic. This property, used in [1] to prove the existence of a first-order graph property that is hard to test, is also investigated in [7]. Ignoring again the running time and concentrating on the number of queries, there seem to be an $\tilde{\Omega}(n)$ lower bound and an upper bound of $n^\alpha$ queries for some $1 \leq \alpha < 2$ for this problem.

**Narrowing the gap between $U_\epsilon$ and $L_\epsilon$.** There is a huge gap between the lower bound function and the upper bound function, on account of the lower bound proof using Szemerédi's regularity lemma. In fact, $U_\epsilon(t)$ is a tower in a polynomial of

$L_\epsilon(t)$ (when comparing their values for the same $t$). It would be interesting to prove a better lower bound using a weaker version of Szemerédi's regularity lemma.

**The gap between the approximation parameters.** In Theorem 2.1, the $\epsilon$-testing number is compared from above with the $\epsilon/3$-approximate algebra number, and from below it is compared with the $3\epsilon$-approximate algebra number. In the upper bound the approximation parameter can be made closer to $\epsilon/2$, and in the lower bound it can be made closer to $\epsilon$, but it would still be interesting to be able to make all approximation parameters arbitrarily close to each other (with the closeness parameter being an additional parameter of the bounding functions).

**Back to boolean functions.** The main question treated in this paper was motivated by a question about boolean functions, so now the original question motivates the following: Is there any analogue of Szemerédi's regularity lemma that can be used to prove lower bounds on testing that a function $f : \{0,1\}^n \to \{0,1\}$ is identical to a given function $g$ up to a permutation of the variables?

There are indications that this question is rather hard. For example, it is not yet entirely clear what the complexity measure for boolean functions should be. Being dependent on a small number of variables cannot be the sole parameter of simplicity, because functions like the parity function on all $n$ variables admit an efficient test despite not being close to any function depending on a small number of variables.

**Acknowledgment.** I wish to thank an anonymous referee for many useful suggestions and comments.

## REFERENCES

[1] N. Alon, E. Fischer, M. Krivelevich, and M. Szegedy, *Efficient testing of large graphs*, Combinatorica, 20 (2000), pp. 451–476.

[2] N. Alon and J. H. Spencer, *The Probabilistic Method*, Wiley-Intersci. Ser. Discrete Math. Optim., 2nd ed., Wiley-Interscience (John Wiley & Sons), New York, 2000.

[3] M. Blum, M. Luby, and R. Rubinfeld, *Self-testing/correcting with applications to numerical problems*, J. Comput. System Sci., 47 (1993), pp. 549–595. A preliminary version appeared in Proceedings of the 22nd Annual ACM Symposium on the Theory of Computing (STOC), 1990.

[4] E. Fischer, *Testing graphs for colorability properties*, Random Structures Algorithms, 26 (2005), pp. 289–309.

[5] E. Fischer, *The art of uninformed decisions: A primer to property testing*, Bull. Eur. Assoc. Theor. Comput. Sci. EATCS, 75 (2001), pp. 97–126.

[6] E. Fischer, G. Kindler, D. Ron, S. Safra, and A. Samorodnitsky, *Testing juntas*, J. Comput. System Sci., 68 (2004), pp. 753–787.

[7] E. Fischer and A. Matzliah, *Testing Graphic Isomorphism*, manuscript, Technion – Israel Institute of Technology, Haifa, Israel, 2005.

[8] E. Fischer and I. Newman, *Testing versus estimation of graph properties*, in Proceedings of the 37th Annual ACM Symposium on the Theory of Computing (STOC), to appear.

[9] O. Goldreich, S. Goldwasser, and D. Ron, *Property testing and its connection to learning and approximation*, J. ACM, 45 (1998), pp. 653–750. A preliminary version appeared in Proceedings of the 37th FOCS, 1996.

[10] O. Goldreich and L. Trevisan, *Three theorems regarding testing graph properties*, Random Structures Algorithms, 23 (2003), pp. 23–57.

[11] D. Ron, *Property testing*, in Handbook of Randomized Computing, Vols. I, II, Comb. Optim. 9, S. Rajasekaran, P. M. Pardalos, J. H. Reif, and J. D. P. Rolim, eds., Kluwer Academic Publishers, Dordrecht, The Netherlands, 2001, pp. 597–649.

[12] R. Rubinfeld and M. Sudan, *Robust characterizations of polynomials with applications to program testing*, SIAM J. Comput., 25 (1996), pp. 252–271. First appeared as a technical report, Cornell University, 1993.

[13] E. Szemerédi, *Regular partitions of graphs*, in Proceedings of the Colloque International CNRS, J. C. Bermond, J. C. Fournier, M. Las Vergnas, and D. Sotteau, eds., Colloq. Internat. CNRS 260, CNRS, Paris, 1978, pp. 399–401.

SIAM J. COMPUT. © 2005 Society for Industrial and Applied Mathematics
Vol. 34, No. 5, pp. 1159–1175

# FAIR SIMULATION RELATIONS, PARITY GAMES, AND STATE SPACE REDUCTION FOR BÜCHI AUTOMATA*

KOUSHA ETESSAMI†, THOMAS WILKE‡, AND REBECCA A. SCHULLER§

**Abstract.** We give efficient algorithms, improving optimal known bounds, for computing a variety of simulation relations on the state space of a Büchi automaton. Our algorithms are derived via a unified and simple parity-game framework. This framework incorporates previously studied notions like *fair* and *direct* simulation, but also a new natural notion of simulation called *delayed* simulation, which we introduce for the purpose of state space reduction. We show that delayed simulation—unlike fair simulation—preserves the automaton language upon quotienting and allows substantially better state space reduction than direct simulation.

Using our parity-game approach, which relies on an algorithm by Jurdziński, we give efficient algorithms for computing all of the above simulations. In particular, we obtain an $O(mn^3)$-time and $O(mn)$-space algorithm for computing both the delayed and the fair simulation relations. The best prior algorithm for fair simulation requires time and space $O(n^6)$. Our framework also allows one to compute bisimulations: we compute the fair bisimulation relation in $O(mn^3)$ time and $O(mn)$ space, whereas the best prior algorithm for fair bisimulation requires time and space $O(n^{10})$.

**Key words.** fair simulation relations, parity games, state space reduction, Büchi automata

**AMS subject classification.** 68Q45

**DOI.** 10.1137/S0097539703420675

## 1. Introduction.

There are at least two distinct purposes for which it is useful to compute simulation relationships between the states of automata: (1) to efficiently establish language containment among nondeterministic automata; and (2) to reduce the state space of an automaton by obtaining its quotient with respect to the equivalence relation underlying the simulation preorder.

For state machines without acceptance conditions, there is a well-understood notion of simulation with a long history (see, e.g., [20, 16]), mainly aimed at comparing the branching behavior of such machines (rather than just their sets of traces). For $\omega$-automata, where acceptance (fairness) conditions are present, there are a variety of different simulation notions (see, e.g., [14, 11]). At a minimum, for such a simulation to be of use for purpose (1), it must have the following property:

(*) whenever state $q'$ "simulates" state $q$, the language of the automaton with start state $q'$ contains the language of the automaton with start state $q$.

As we will see in section 5, however, this property alone is not sufficient to assure usefulness for purpose (2), which requires the following stronger property:

(**) the "simulation quotient" preserves the language of the automaton.

We will state precisely what is meant by a simulation quotient later.

In [14], a number of different simulation notions for $\omega$-automata were studied using a game-theoretic framework. The authors also introduced a new natural notion of simulation, titled *fair* simulation. They showed how to compute fair simulations for both Büchi and, more generally, Streett automata. For Büchi automata, their algorithm requires $O(n^6)$ time to determine, for one pair of states $(q, q')$, whether $q'$

*Received by the editors January 6, 2003; accepted for publication (in revised form) August 18, 2004; published electronically June 3, 2005. This paper is based on the conference paper [9].
http://www.siam.org/journals/sicomp/34-5/42067.html

†University of Edinburgh, Edinburgh, Scotland EH8 9YL (kousha@inf.ed.ac.uk).

‡Christian-Albrechts-Universität, 24098 Kiel, Germany (wilke@ti.informatik.uni-kiel.de).

§Cornell University, Ithaca, NY 14853 (reba@math.cornell.edu).

fairly simulates $q$. Their algorithm relies on an algorithm for tree automaton emptiness testing developed in [19]. In this paper, we present a new comparatively simple algorithm for Büchi automata. Our algorithm reduces the problem to a parity game computation, for which we use a recent elegant algorithm by Jurdziński [17], along with some added enhancements to achieve our bounds. Our algorithm determines in time $O(mn^3)$ and space $O(mn)$ *all* such pairs $(q, q')$ of states in an input automaton $A$ where $q'$ simulates $q$. Here $m$ denotes the number of transitions and $n$ the number of states of $A$. In other words, our algorithm computes the entire maximal fair simulation relation on the state space in the stated time and space bound.

In [14], the authors were interested in using fair simulation for purpose (1) and thus did not consider quotients with respect to fair simulation. The question arises whether fair simulation can be used for purpose (2), i.e., whether it satisfies property (**). The answer is no: we show that quotienting with respect to fair simulation fails badly at preserving the underlying language, under any reasonable definition of a quotient. (Closely related observations were also made in [15].) On the other hand, there is an obvious and well-known way to define simulation so that quotients do preserve the underlying language: *direct* simulation[1] [6] simply accommodates acceptance into the standard definition of simulation [20] by asserting that only an accept state can simulate another accept state. Direct simulation has already been used extensively (see, e.g., [8, 22]) to reduce the state space of automata. See also [5], where simulation minimization for ordinary Kripke structures was studied. Both [8] and [22] describe tools for optimized translations from linear temporal logic to automata, where one of the key optimizations is simulation reduction. However, as noted in [8], direct simulation alone is not able to reduce many obviously redundant state spaces. Recall that, in general, it is PSPACE-hard to find the minimum equivalent automaton for a given nondeterministic automaton. Thus, there is a need for efficient algorithms and heuristics that reduce the state space substantially.

We introduce a natural intermediate notion between direct and fair simulation, called *delayed* simulation, which satisfies property (**). We show that delayed simulation can yield substantially greater reduction—by an arbitrarily large factor—than direct simulation. We provide an algorithm for computing the entire delayed simulation relation which arises from precisely the same parity-game framework and has the same complexity as our algorithm for fair simulation.

Last, our parity-game framework also easily accommodates computation of bisimulation relations (which are generally less coarse than simulation). In particular, we show that the fair bisimulation relation on Büchi automata can be computed in time $O(mn^3)$ and space $O(mn)$. Fair bisimulation was studied in [15] for Büchi and Streett automata, where for Büchi automata they gave an $O(n^{10})$–time and space algorithm to compute whether one state is fair bisimilar to another.

This paper is based on the conference paper [9]. Several papers have since appeared that build on and/or use our work: [13, 10, 7, 4]. Independent of [9], in [3] Bustan and Grumberg obtained an algorithm for computing fair simulation which, while it did not improve the $O(n^6)$-time complexity of [14], improved the space complexity to $O(mn)$. The algorithms described here have been implemented in the TMP tool, available for download at http://www1.bell-labs.com/project/TMP/.

The paper is organized as follows: in section 2, we define all (bi)simulation notions used in the paper. In section 3 we show how for each simulation notion (and fair bisimulation), given a Büchi automaton, we can define a parity game that captures

[1] Direct simulation is called *strong* simulation in [8].

the (bi)simulation. In section 4, we use our variant of Jurdziński's algorithm for parity games to give efficient algorithms for computing several such (bi)simulation relations. In section 5, we prove that the delayed simulation quotient can be used to reduce automaton size, and yields better reduction than direct simulation, but that the fair simulation quotient cannot be so used. We conclude in section 6.

For background on simulation and its versions incorporating acceptance, see, e.g., [20, 16] and [14], respectively. For background on Büchi automata and automata on infinite words in general, see [23, 24, 12].

We thank an anonymous referee for pointing out that an earlier version of the algorithm in Figure 2 was too complicated.

**2. Simulation and bisimulation relations.** We now define various notions of simulation, including fair and the new delayed simulation, in terms of appropriate games.

**2.1. Büchi automata.** As usual, a *Büchi automaton* $A = \langle \Sigma, Q, q_I, \Delta, F \rangle$ has an alphabet $\Sigma$, a state set $Q$, an initial state $q_I \in Q$, a transition relation $\Delta \subseteq Q \times \Sigma \times Q$, and a set of final states $F \subseteq Q$. We will henceforth assume that the automaton has no *dead ends*; i.e., from each state of $A$ there is a path of length at least 1 to *some* state in $F$. Unless the automaton is trivial (i.e., has an empty $\omega$-language), it is easy to make sure this property holds without changing the accepting runs from any state, using a simple search to eliminate unnecessary states and transitions. (Also, it is easy to check nontriviality while doing the same search. The running time of the search is linear in the size of the automaton.)

Recall that a *run* of $A$ is a sequence $\pi = q_0 a_0 q_1 a_1 q_2 \ldots$ of states alternating with letters such that for all $i$, $(q_i, a_i, q_{i+1}) \in \Delta$. The $\omega$-word associated with $\pi$ is $w_\pi = a_0 a_1 a_2 \ldots$ The run $\pi$ is *initial* if it starts with $q_I$; it is *accepting* if there exist infinitely many $i$ with $q_i \in F$. The language defined by $A$ is $L(A) = \{w_\pi \in \Sigma^\omega \mid \pi \text{ is an initial, accepting run of } A\}$. We may want to change the start state of $A$ to a different state $q$; the revised automaton is denoted by $A[q]$.

**2.2. Simulation relations.** As in [14], we define simulation game-theoretically. We will focus on simulations between distinct states of the same automaton ("autosimulations"), because we are primarily interested in state space reduction. Simulations between different automata can be treated by considering autosimulations between the states of the automaton consisting of their disjoint union. In [14], the authors presented their work in terms of Kripke structures with fairness constraints. We use Büchi automata directly, where labels are on transitions instead of states. This difference is inconsequential for our results.

We will consider four kinds of simulations: ordinary simulation, which ignores acceptance, as well as three variants which incorporate acceptance conditions of the given automaton, in particular, our new delayed simulation. All four simulations are based on the same game, which is described first. They differ only in the winning condition.

Let $A$ be a Büchi automaton as above and $q_0$ and $q'_0$ arbitrary states of $A$. The *basic game* $G_A(q_0, q'_0)$ is played by two players, *Spoiler* and *Duplicator,* in rounds, where, at the beginning and at the end of each round, two pebbles, *Red* and *Blue*, are placed on two states (possibly the same). At the start, round 0, Red and Blue are placed on $q_0$ and $q'_0$, respectively. Assume that, at the beginning of round $i$, Red is on state $q_i$ and Blue is on $q'_i$. Then:

1. Spoiler chooses a transition $(q_i, a, q_{i+1}) \in \Delta$ and moves Red to $q_{i+1}$.

2. Duplicator, responding, must choose a transition $(q'_i, a, q'_{i+1}) \in \Delta$ and moves Blue to $q'_{i+1}$. If no $a$-transition starting from $q'_i$ exists, then the game halts and Spoiler wins.

Either the game halts, in which case Spoiler wins, or the game produces two infinite runs, $\pi = q_0 a_0 q_1 a_1 q_2 \dots$ and $\pi' = q'_0 a_0 q'_1 a_1 q'_2 \dots$, built from the transitions visited by the two pebbles (and such that the same word is associated with them). The pair $(\pi, \pi')$ is called an *outcome* of the game. Given such an outcome, the following rules are used to determine the winner.

DEFINITION 1 (simulation games). *Let $A$ be a Büchi automaton, let $(q_0, q'_0) \in Q^2$, and let $(\pi, \pi')$ be an outcome of $G_A(q_0, q'_0)$ with $\pi = q_0 a_0 q_1 a_1 q_2 \dots$ and $\pi' = q'_0 a_0 q'_1 a_1 q'_2 \dots$*

1. *The* ordinary simulation game, *denoted by $G^o_A(q_0, q'_0)$, is the basic game $G_A(q_0, q'_0)$ extended by the rule that the outcome $(\pi, \pi')$ is winning for Duplicator (i.e., as long as the game does not halt, Duplicator wins).*

2. *The* direct (strong) simulation game, *denoted by $G^{di}_A(q_0, q'_0)$, is the basic game $G_A(q_0, q'_0)$ extended by the rule that the outcome $(\pi, \pi')$ is winning for Duplicator iff, for all $i$, if $q_i \in F$, then also $q'_i \in F$.*

3. *The* delayed simulation game, *denoted by $G^{de}_A(q_0, q'_0)$, is the basic game $G_A(q_0, q'_0)$ extended by the rule that the outcome $(\pi, \pi')$ is winning for Duplicator iff, for all $i$, if $q_i \in F$, then there exists $j \geq i$ such that $q'_j \in F$.*

4. *The* fair simulation game, *denoted by $G^f_A(q_0, q'_0)$, is the basic game $G_A(q_0, q'_0)$ extended by the rule that the outcome $(\pi, \pi')$ is winning for Duplicator iff there are infinitely many $j$ such that $q'_j \in F$ or there are only finitely many $i$ such that $q_i \in F$. In all other cases, Spoiler wins.*

In other words, in ordinary simulation games, fairness conditions are ignored; Duplicator wins as long as the game does not halt. And in fair simulation games, Duplicator's winning condition is as follows: if there are infinitely many $i$ such that $q_i \in F$, then there are also infinitely many $j$ such that $q'_j \in F$.

*Remark* 1. Let $A$ be a Büchi automaton and $\star \in \{di, de, f\}$. If $(\pi, \pi')$ is the outcome of a play of $G^\star_A(q, q')$ which Duplicator wins, then $\pi'$ is accepting if $\pi$ is.

Let $\star \in \{o, di, de, f\}$. A *strategy* for Duplicator in game $G^\star_A(q_0, q'_0)$ is a partial function $f\colon Q(Q\Sigma Q)^* \to Q$ which, given the history of the game up to a certain point, determines the next move of Duplicator. Formally, $f$ is a strategy for Duplicator if $f(q_0) = q'_0$ and $(q'_i, a_i, q'_{i+1}) \in \Delta$ holds for every sequence $q_0 q'_0 a_0 q_1 q'_1 a_1 \dots a_i q_{i+1}$ with $(q_j, a_j, q_{j+1}) \in \Delta$ and $q'_j = f(q_0 q'_0 a_0 \dots a_j q_j)$ for $j \leq i$. Observe that the existence of a strategy implies that Duplicator has a way of playing such that the game does not halt. A strategy $f$ for Duplicator is a *winning* strategy if, no matter how Spoiler plays, Duplicator always wins. Formally, a strategy $f$ for Duplicator is winning if whenever $\pi = q_0 a_0 q_1 a_1 \dots$ is a run through $A$ and $\pi' = q'_0 a_0 q'_1 a_1 q'_2 \dots$ is the run defined by

$$q'_{i+1} = f(q_0 q'_0 a_0 q_1 q'_1 a_1 \dots q_{i+1}), \tag{1}$$

then $(\pi, \pi')$ is winning for Duplicator (as specified in Definition 1). Observe that $\pi'$ is well-defined.

DEFINITION 2 (simulation relations). *Let $A$ be a Büchi automaton. A state $q'$* ordinary, direct, delayed, fair simulates *a state $q$ if there is a winning strategy for Duplicator in $G^\star_A(q, q')$ where $\star = o$, $di$, $de$, or $f$, respectively. We denote such a relationship by $q \preceq_\star q'$ (where $A$ is implicit).*

Our game definition of fair simulation deviates very slightly from that given in [14], but is equivalent since we consider only automata with no dead ends.

We first prove fundamental properties of the defined simulation relations.

PROPOSITION 3. *Let $A$ be a Büchi automaton.*

1. *For $\star \in \{o, di, de, f\}$, $\preceq_\star$ is a reflexive, transitive relation (also called* pre-order *or* quasi-order*) on the state set $Q$.*
2. *The relations are ordered by containment: $\preceq_{di} \subseteq \preceq_{de} \subseteq \preceq_f \subseteq \preceq_o$.*
3. *For $\star \in \{di, de, f\}$, if $q \preceq_\star q'$, then $L(A[q]) \subseteq L(A[q'])$.*

*Proof.* Reflexivity is obvious, as is part 2. To prove transitivity, suppose that $q_0 \preceq_\star q'_0 \preceq_\star q''_0$ for some $\star \in \{o, di, de, f\}$. Then, by definition, Duplicator has winning strategies in both $G^\star_A(q_0, q'_0)$ and $G^\star_A(q'_0, q''_0)$, say $f$ and $f'$. We combine these to get a winning strategy $f''$ for Duplicator in the game $G^\star_A(q_0, q''_0)$ as follows. If $f(q_0q'_0a_0q_1q'_1a_1 \dots q_{i+1}) = q'_{i+1}$ and $f'(q'_0q''_0a_0q'_1q''_1a_1 \dots q'_{i+1}) = q''_{i+1}$, we set $f(q_0q''_0a_0q_1q''_1a_1 \dots q_{i+1}) = q''_{i+1}$. It is easy to see that this defines a strategy for Duplicator. To see that $f''$ is in fact winning, let $\pi = q_0a_0q_1a_1 \dots$ be a run through $A$ and let $\pi'' = q''_0a_0q''_1a_1 \dots$ be the run defined by

$$q''_{i+1} = f''(q_0q''_0a_0q_1q''_1a_1 \dots q_{i+1}). \tag{2}$$

We need to argue that $(\pi, \pi'')$ is winning for Duplicator. By induction, one easily proves that if $\pi' = q'_0a_0q'_1a_1 \dots$ is defined by (1), then

$$q''_{i+1} = f'(q'_0q''_0a_0q'_1q''_1a_1 \dots q'_{i+1}). \tag{3}$$

This means that $(\pi, \pi')$ is winning for Duplicator in $G^\star_A(q_0, q'_0)$ and $(\pi', \pi'')$ is winning for Duplicator in $G^\star_A(q'_0, q''_0)$. For instance, when $\star = de$, this implies the following: if $q_i \in F$, there exists $j \geq i$ such that $q'_j \in F$, which, in turn, means there exists $k \geq j$ such that $q''_k \in F$. That is, $(\pi, \pi'')$ is winning for Duplicator in $G^{de}_A(q_0, q''_0)$. Similar arguments apply in the other cases.

To prove part 3, assume $\star \in \{di, de, f\}$, $q_0 \preceq_\star q'_0$, and $w \in L(A[q_0])$ with $w = a_0a_1 \dots$. Then there exists a winning strategy $f$ for Duplicator in $G^\star_A(q_0, q'_0)$ and an accepting run $\pi = q_0a_0q_1a_1 \dots$ of $A$ starting with $q_0$. Imagine Spoiler plays in $G^\star_A(q_0, q'_0)$ just as $\pi$ prescribes this and Duplicator replies according to $f$. Then a run $\pi' = q'_0a_0q'_1a_1 \dots$ of $A$ is built up according to (1). Since $\pi$ is accepting and $f$ is winning, $\pi'$ will also be accepting; see Remark 1. ◻

Thus, delayed simulation is a new notion of intermediate "coarseness" between direct and fair simulation. We will see in section 5 why it is more useful for state space reduction.

**2.3. Bisimulation relations.** For all the mentioned simulations there are corresponding notions of bisimulation, defined via a modification of the game. We will not provide detailed definitions for bisimulation; instead we describe intuitively the simple needed modifications.

The bisimulation game differs from the simulation game in that Spoiler gets to choose in each round which of the two pebbles, Red or Blue, to move, and Duplicator has to respond with a move of the other pebble.

The winner of the game is determined very similarly: if the game comes to a halt, Spoiler wins. If not, the winning condition for *fair* bisimulation (see [15]) is as follows: "if an accept state appears infinitely often on one of the two runs $\pi$ and $\pi'$, then an accept state must appear infinitely often on the other as well." The winning condition for *delayed* bisimulation is as follows: "if an accept state is seen at position $i$ of either run, then an accept state must be seen thereafter at some position $j \geq i$ of the other run." The winning condition for *direct* bisimulation becomes "if an accept state is seen at position $i$ of either run, it must be seen at position $i$ of both runs."

Strategies and winning strategies for the bisimulation games are defined similarly. Note, however, that the definitions have to take into account that Spoiler may choose his pebble.

Bisimulations define an equivalence relation $\approx_\star^{bi}$ (not only a preorder) on the state space, and the following containments hold: $\approx_{di}^{bi} \subseteq \approx_{de}^{bi} \subseteq \approx_f^{bi} \subseteq \approx_o^{bi}$. Generally, bisimulation is less coarse than the equivalence derived from the simulation preorder, which we describe in section 5, i.e., $\approx_\star^{bi} \subseteq \approx_\star$.

## 3. Reformulating simulations and bisimulations as parity games.

**3.1. Simulation.** We now show how, given a Büchi automaton $A$ and $\star \in \{o, di, de, f\}$, we can obtain in a straightforward way a parity-game graph $G_A^\star$ such that the winning vertices in $G_A^\star$ for Even (a.k.a. Player 0) in the parity game determine precisely the pairs of states $(q, q')$ of $A$ where $q'$ $\star$-simulates $q$. Importantly, the size of these parity-game graphs will be $O(|Q||\Delta|)$, and the nodes of the game graphs will be labeled by at most three distinct "priorities." In fact, only one priority will suffice for $G_A^o$ and $G_A^{di}$, while $G_A^{de}$ and $G_A^f$ will use three priorities.

We briefly review here the basic formulation of a parity game. A parity-game graph $G = \langle V_0, V_1, E, p\rangle$ has two disjoint sets of vertices, $V_0$ and $V_1$, whose union is denoted $V$. There is an edge set $E \subseteq V \times V$, and $p\colon V \to \{0, \ldots, d-1\}$ is a mapping that assigns a *priority* to each vertex.

A parity game on $G$, starting at vertex $v_0 \in V$, is denoted $\mathrm{P}(G, v_0)$, and is played by two players, *Even* and *Odd*. The play starts by placing a pebble on vertex $v_0$. Thereafter, the pebble is moved according to the following rule: with the pebble currently on a vertex $v_i$, and $v_i \in V_0$ ($V_1$), Even (Odd, respectively) plays and moves the pebble to a neighbor $v_{i+1}$, that is, such that $(v_i, v_{i+1}) \in E$.

If ever the above rule cannot be applied, i.e., someone can't move because there are no outgoing edges, the game ends, and the player who cannot move loses. Otherwise, the game goes on forever and defines a path $\pi = v_0 v_1 v_2 \ldots$ in $G$, called a *play* of the game. The winner of the play is determined as follows. Let $k_\pi$ be the minimum priority that occurs infinitely often in the play $\pi$, i.e., so that for infinitely many $i$, $p(v_i) = k_\pi$ and $k_\pi$ is the least number with this property. Even wins if $k_\pi$ is even, whereas Odd wins if $k_\pi$ is odd.

We now show how to build the game graphs $G_A^\star$. All the game graphs are built following the same general pattern, with some minor alterations. We start with $G_A^f$. The game graph $G_A^f = \langle V_0^f, V_1^f, E_A^f, p_A^f\rangle$ will have three priorities (i.e., the range of $p_A^f$ will be $\{0, 1, 2\}$). For each pair of states $(q, q') \in Q^2$, there will be a vertex $v_{(q,q')} \in V_0^f$ such that Even has a winning strategy from $v_{(q,q')}$ iff $q'$ fair simulates $q$. Formally, $G_A^f$ is defined by

$$V_0^f = \{v_{(q,q',a)} \mid q, q' \in Q \wedge \exists q''((q'', a, q) \in \Delta)\}, \tag{4}$$

$$V_1^f = \{v_{(q,q')} \mid q, q' \in Q\}, \tag{5}$$

$$\begin{aligned} E_A^f = {} & \{(v_{(q_1,q_1',a)}, v_{(q_1,q_2')}) \mid (q_1', a, q_2') \in \Delta\} \\ & \cup \{(v_{(q_1,q_1')}, v_{(q_2,q_1',a)}) \mid (q_1, a, q_2) \in \Delta\}, \end{aligned} \tag{6}$$

$$p_A^f(v) = \begin{cases} 0 & \text{if } v = v_{(q,q')} \text{ and } q' \in F, \\ 1 & \text{if } v = v_{(q,q')},\ q \in F, \text{ and } q' \notin F, \\ 2 & \text{otherwise.} \end{cases} \tag{7}$$

Let's first explain the underlying idea. The parity game mimics the simulation game. Even takes over the role of Duplicator and Odd takes over the role of Spoiler: when in the parity game the current position is node $v_{(q,q')}$, it denotes the situation in the simulation game when the red pebble is on $q$, the blue pebble is on $q'$, and it is Spoiler's turn to move; $v_{(q,q',a)}$ denotes the situation where the red pebble is on $q$, the blue pebble is on $q'$, it is Duplicator's turn to move, and the last transition taken by Spoiler was labeled by $a$. The priority function is defined in such a way that every time Duplicator visits a final state, the priority function returns 0. It returns only 1 if Spoiler visits a final state, but Duplicator does not. In all other cases, 2 is returned. That is, Spoiler wins iff he visits an accept state infinitely often but Duplicator does not. This is exactly what is needed.

We now describe how $G_A^f$ can be modified to obtain $G_A^o$ and $G_A^{di}$, both of which require only trivial modification to $G_A^f$. The parity-game graph $G_A^o$ is exactly the same as $G_A^f$, except that all nodes will receive priority 0, i.e., $p_A^o(v) = 0$ for all $v$. This reflects the winning condition of the ordinary simulation game.

The parity-game graph $G_A^{di}$ is just like $G_A^o$, meaning every vertex has priority 0, but some edges (the ones into and out of states of the form $v_{(q,q')}$ where $q \in F$ but $q' \notin F$) are eliminated in order to take care of the winning condition of the direct simulation game:

$$E_A^{di} = E_A^f \setminus (\{(v, v_{(q,q')}) \mid q \in F \wedge q' \notin F\} \cup \{(v_{(q,q')}, w) \mid q \in F \wedge q' \notin F\}). \tag{8}$$

Finally, to define $G_A^{de}$ we need to modify the game graph somewhat more. For each vertex of $G_A^f$ there will be at most two corresponding vertices in $G_A^{de}$:

$$V_0^{de} = \{v_{(b,q,q',a)} \mid q, q' \in Q \wedge b \in \{0,1\} \wedge \exists q''((q'', a, q) \in \Delta)\}, \tag{9}$$

$$V_1^{de} = \{v_{(b,q,q')} \mid q, q' \in Q \wedge b \in \{0,1\} \wedge (q' \in F \rightarrow b = 0)\}. \tag{10}$$

The extra bit $b$ encodes whether or not, thus far in the simulation game, the red pebble has witnessed an accept state without the blue pebble having witnessed one since then. The edges of $G_A^{de}$ are as follows:

$$\begin{aligned} E_A^{de} = \{&(v_{(b,q_1,q_1',a)}, v_{(b,q_1,q_2')}) \mid (q_1', a, q_2') \in \Delta \wedge q_2' \notin F\} \\ \cup \{&(v_{(b,q_1,q_1',a)}, v_{(0,q_1,q_2')}) \mid (q_1', a, q_2') \in \Delta \wedge q_2' \in F\} \\ \cup \{&(v_{(b,q_1,q_1')}, v_{(b,q_2,q_1',a)}) \mid (q_1, a, q_2) \in \Delta \wedge q_2 \notin F\} \\ \cup \{&(v_{(b,q_1,q_1')}, v_{(1,q_2,q_1',a)}) \mid (q_1, a, q_2) \in \Delta \wedge q_2 \in F\}. \end{aligned} \tag{11}$$

Last, we describe the priority function of $G_A^{de}$:

$$p_A^{de}(v) = \begin{cases} b & \text{if } v = v_{(b,q,q')}, \\ 2 & \text{if } v \in V_0^{de}. \end{cases} \tag{12}$$

In other words, we will assign priority 1 to only those vertices in $V_1$ that signify that an "unmatched" accept has been visited by the red pebble.[2] The priority function

[2]Note that it is possible to use only two priorities in $p_A^{de}$ by assigning a vertex $v$ the priority $b$, where $b$ is the indicator bit of $v$. However, it turns out that using two priorities is a disadvantage over three because the encoding would not have property 3 of Lemma 4, which we need for our complexity bounds.

makes sure that the smallest number occurring infinitely often is 1 iff from some point onwards the bit in the first component is 1. Now observe that this bit is 1 iff a final state has been visited by Spoiler but not yet matched by Duplicator. In this way the winning condition of the delayed simulation game is transferred to the parity game.

The following lemma gathers a collection of facts we will need.

LEMMA 4. *Let $A$ be a Büchi automaton.*

1. *For $\star \in \{o, di, f\}$, Even has a winning strategy in $\mathrm{P}(G^{\star}_A, v_{(q_0,q_0')})$ iff $q_0'$ $\star$-simulates $q_0$ in $A$.*
*For $\star = de$, this statement holds if $v_{(q_0,q_0')}$ is replaced by $v_{(b,q_0,q_0')}$, letting $b = 1$ if $q_0 \in F$ and $q_0' \notin F$, and $b = 0$ otherwise.*
2. *For $\star \in \{o, di, de, f\}$, $|G^{\star}_A| \in O(|\Delta||Q|)$, i.e., the number of vertices and the number of edges is $O(|\Delta||Q|)$.*
3. *For $\star \in \{f, de\}$, $|\{v \in V^{\star}_A \mid p^{\star}_A(v) = 1\}| \in O(|Q|^2)$.*

*Proof.* Part 1 is obvious from the explanations given above.

To prove part 2, first consider the case where $\star \in \{f, o\}$. In this case, $V^{\star}_1$ contains exactly $|Q|^2$ vertices, and since by assumption every state of $A$ has a transition leaving it, $|Q|^2 \leq |\Delta||Q|$. Similarly, $V^{\star}_0$ has, for every $q'$, a state $v_{(q,q',a)}$ iff there is a transition to $q$ labeled by $a$. Thus $|V_0| \leq |\Delta||Q|$.

As far as $|E^f_A|$, for every transition $(q, a, q') \in \Delta$, and every $q'' \in Q$, there is an edge $(v_{(q,q'')}, v_{(q',q'',a)}) \in E^{\star}$. There are $|\Delta||Q|$ such edges. Likewise, there are $\leq |\Delta||Q|$ edges from $V^{\star}_0$ to $V^{\star}_1$. So $|E^f_A| \in O(|\Delta||Q|)$. Thus, the size of $G^{\star}_A$ is $O(|\Delta||Q|)$. Now observe that if $\star = o$, the vertices do not change and the edge set is a subset, and if $\star = de$, the number of vertices and edges is larger by at most a factor of 2.

Last, since the vertices labeled by priority 1 in both $G^f$ and $G^{de}$ are a subset of $V_0$, clearly $|p^{-1}(1)| \in O(|Q|^2)$. □

Since vertices of $G^o_A$ and $G^{di}_A$ only get assigned a single priority, we can dispense with algorithms for computing ordinary and direct simulation right away, matching the best known upper bounds:

PROPOSITION 5 (see [16, 2]). *Given a Büchi automaton $A$, with $n$ states and $m$ transitions, both $\preceq_o$ and $\preceq_{di}$ can be computed in time and space $O(mn)$.*

*Proof.* $G^{\star}_A$ here has size $O(|\Delta||Q|)$ and only one priority. For such game graphs, we can compute the winning set for Even using a variant of AND/OR graph accessibility, which can be computed in linear time (see, e.g., [1]). The only vertices in the game graph that have no outgoing edges are in $V_0$. These are losing positions for Even, as are any other vertices from which these are accessible in the and/or sense (vertices from $V_0$ are considered "and nodes" and vertices from $V_1$ are considered "or nodes"). All the remaining vertices are winning positions for Even. □

Algorithms for computing the other simulation relations will be presented in section 4.

## 3.2. Bisimulation.

For $\star \in \{o, di, de, f\}$, $\star$-bisimulations can also be reformulated as parity games. For improving the complexity, such a reformulation helps only for fair bisimulation. Ordinary and direct bisimulation have known $O(m \log n)$-time algorithms (see [21]), and we will see that delayed bisimulation corresponds to direct bisimulation after some linear-time preprocessing on accept states of the Büchi automaton.

We formulate fair bisimulation with a parity-game graph $G^{f\,bi}_A$ as follows. The vertex sets of $G_A^{f\,bi}$ are

$$V_0^{f\,bi} = \{v_{(q,q',a,b_1,b_2)} \mid q, q' \in Q \wedge b_1, b_2 \in \{0,1\} \wedge \exists q''((q'', a, q) \in \Delta)\}, \tag{13}$$

$$V_1^{f\,bi} = \{v_{(q_0,q_1,b_2)} \mid q, q' \in Q \wedge b_2 \in \{0,1\}\}. \tag{14}$$

The two bits $b_1$ and $b_2$ will encode (1) which pebble was moved by Spoiler in this round, and (2) which of the two pebbles was latest to visit an accept state (prior to this round and with precedence for the red pebble, with 0 encoding the red pebble). For $q_0, q_1 \in Q$ and $b_2 \in \{0,1\}$, let

$$\text{new}(q_0, q_1, b_2) = \begin{cases} 0 & \text{if } q_0 \in F, \\ 1 & \text{if } q_0 \notin F \text{ and } q_1 \in F, \\ b_2 & \text{otherwise.} \end{cases} \tag{15}$$

The edge set $E_A^{f\,bi}$ is the union of

$$\{(v_{(q_0,q_1,b_2)}, v_{(q'_0,q'_1,a,b_1,\text{new}(q_0,q_1,b_2))}) \mid (q_{b_1}, a, q'_{b_1}) \in \Delta \wedge q_{1-b_1} = q'_{1-b_1}\} \tag{16}$$

and

$$\{(v_{(q_0,q_1,a,b_1,b_2)}, v_{(q'_0,q'_1,b_2)}) \mid (q_{1-b_1}, a, q'_{1-b_1}) \in \Delta \wedge q_{b_1} = q'_{b_1}\}. \tag{17}$$

The priority of a vertex is determined using the following function. For $q_0, q_1, b$ let

$$\text{pr}(q_0, q_1, b) = \begin{cases} 0 & \text{if } q_{1-b} \in F, \\ 1 & \text{if } q_{1-b} \notin F \text{ and } q_b \in F, \\ 2 & \text{otherwise.} \end{cases} \tag{18}$$

For $v \in V_0$, $p_A^{f\,bi}(v) = 2$, and for $v_{(q_0,q_1,b_2)} \in V_1$,

$$p_A^{f\,bi}(v_{(q_0,q_1,b_2)}) = \text{pr}(q_0, q_1, b_2). \tag{19}$$

The correspondence of this parity game and fair bisimulation is as follows, similar to Lemma 4.

LEMMA 6. *Let $A$ be a Büchi automaton.*

1. *Even has a winning strategy in* $\text{P}(G_A^{f\,bi}, v_{(q_0,q_1,0)})$ *iff $q_0$ and $q_1$ are fair-bisimilar in $A$.*
2. $|G_A^{f\,bi}| \in O(|\Delta||Q|)$ *and* $|\{v \in V_A^{f\,bi} \mid p_A^{f\,bi}(v) = 1\}| \in O(|Q|^2)$.

*Proof.* It is clear that the parity game models the bisimulation game in a straightforward way as far as the sequence of the visited positions is concerned. We show that the winning condition is also taken care of.

Assume $\pi = q_0a_0q_1a_1\ldots$ and $\pi' = q'_0a_0q'_1a_1\ldots$ are two runs. Let $b_0b_1\ldots$ be the sequence of bits defined by $b_0 = 0$ and $b_{i+1} = \text{new}(q_i, q'_i, b_i)$. Finally, let $p_i$ be defined by $p_i = \text{pr}(q_i, q'_i, b_i)$. This describes exactly what happens in the game. We proceed by a case distinction.

Clearly, if $\pi$ and $\pi'$ are not accepting, then $p_j = 2$ for all $j$ large enough and Even wins, which is required. Next, assume $\pi$ is accepting, and $\pi'$ is not. Then there exists $i$ such that $q'_j \notin F$ for $j \geq i$ and $q_j \in F$ for infinitely many $j$, say $i_0 < i_1 < i_2 < \cdots$ are such that $q_{i_j} \in F$ for every $j$. Assume $i_k > i$. According to the definition of *new*, $b_{i_j} = 0$ for $j > k$. Thus, by definition of *pr*, for every $j > i_k$, $p_j = 1$ if $j = i_l$ for some $l > k$ and $p_j = 2$ otherwise—Even loses. The same argument applies if $\pi$ is not accepting but $\pi'$ is. (The precedence for red does not play any role here.) Finally, assume $\pi$ and $\pi'$ are accepting. Let $i_0 < i_1 < \cdots$ be an infinite sequence such that $q_{i_j} \in F$ for all $j$. For $j$, let $i'_j$ be the least $k > i_j$ such that $q'_k \in F$. We will have $b_k = 0$ for $i_j < k \leq i'_j$, which means $p_{i'_j} = 0$ for every $j$—Even wins.

The claim about the size of $G_A^{f\,bi}$ and the number of its vertices of priority 1 can be proved along the same lines as Lemma 4. □

This enables us to give an efficient algorithm for computing fair bisimulation in section 4.

To compute delayed bisimulation efficiently, we show that the delayed bisimulation relation corresponds to the direct bisimulation relation after some linear-time preprocessing on the accept states of the Büchi automaton. Consider the following closure operation on the set of accept states. Let $\mathrm{cl}(A)$ be the Büchi automaton obtained from $A$ by repeating the following until a fixed point is reached: while there is a state $q$ such that all of its successors are in $F$, put $q$ in $F$. Call the revised set of accept states $F'$. Clearly, $\mathrm{cl}(A)$ can be computed in linear time and $L(A) = L(\mathrm{cl}(A))$.

Proposition 7. *Let $A$ be a Büchi automaton. For any two states $q$ and $q'$, $q \approx_{de}^{bi} q'$ in $A$ iff $q \approx_{di}^{bi} q'$ in* $\mathrm{cl}(A)$.

*Proof.* We show that for every pair $(q, q')$ of states, a winning strategy for Duplicator in the delayed bisimulation game on $(q, q')$ in $A$ is a winning strategy for Duplicator in the direct bisimulation game on $(q, q')$ in $\mathrm{cl}(A)$, and vice versa. By definition of the bisimulation relations, this proves the proposition.

First, let $f$ be a winning strategy for Duplicator in the delayed bisimulation game on $(q_0, q'_0)$ in $A$. Suppose that with Duplicator playing according to strategy $f$ the direct bisimulation game reaches $(q_i, q'_i)$ after $i$ rounds. We have to show that $q_i \in F'$ iff $q'_i \in F'$. Suppose, for contradiction, that $q_i \notin F'$, while $q'_i \in F'$ (the other situation is symmetric). We will show how Spoiler can win the delayed bisimulation game. Since $q_i \notin F'$, there is an infinite path leaving $q_i$ such that no state on this path is an accepting state. Spoiler's strategy is to play this path. Since $q'_i \in F'$, there is no such path (without an accept state on it) starting at $q'_i$. Therefore, regardless of how Duplicator plays, when $(q_0 a_0 q_1 a_1 \ldots, q'_0 a_0 q'_1 a_1 \ldots)$ is the outcome of the play, then $q'_i \in F'$, but $q_j \notin F'$ for all $j \geq i$—Spoiler wins the delayed bisimulation game.

Second, let $f$ be a winning strategy for Duplicator in the direct bisimulation game on $(q, q')$ in $\mathrm{cl}(A)$ and suppose Duplicator plays according to $f$ in the delayed bisimulation game. Let $(q_0 a_0 q_1 a_1 \ldots, q'_0 a_0 q'_1 a_1 \ldots)$ be any outcome of such a play. We have to show that it satisfies Duplicator's winning condition. So let $i$ be any index such that $q_i \in F$. Then, by definition of $F'$, $q_i \in F'$. But since $f$ is a winning strategy in the direct bisimulation game, this implies $q'_i \in F'$. As every infinite path out of $q'_i$ contains an accept state, there must be a $j \geq i$ such that $q'_j \in F$. Symmetrically, if $q'_i \in F$, then there exists $j \geq i$ such that $q_j \in F$. □

Taking into account that direct bisimulation can be computed in time $O(m \log n)$ (see [21]), we conclude with the following result.

Corollary 8. *Delayed bisimulation can be computed in time $O(m \log n)$.*

## 4. Fast parity-game algorithm to compute simulations (and bisimulations) efficiently.

Using the parity-game graphs $G_A^f$, $G_A^{f\,bi}$, and $G_A^{de}$, we give fast algorithms for computing the relations $\preceq_f$, $\approx_f^{bi}$, and $\preceq_{de}$. To this effect, we describe an efficient implementation of an algorithm for solving parity games presented by Jurdziński in [17]. This algorithm uses progress measures (see also [18, 25]) to compute the set of vertices in a parity game from which Even has a winning strategy.

Henceforth, we assume all parity-game graphs have neither self loops nor dead ends. (We can always obtain an "equivalent" such graph in linear time.) We start with some terminology, closely following the notation of [17]. Let $G$ be a parity-game graph as before, $n'$ its number of vertices, $m'$ its number of edges. For computing simulations, we only need assume there are only three priorities, that is, $p\colon V \to \{0,1,2\}$. However, we will present the algorithm in its full generality, i.e., $p\colon V \to \{0,1,\dots,d-1\}$, since the algorithm is of much broader interest.

Let $[n] = \{0,\dots,n-1\}$, and let $n_i = |p^{-1}(i)|$. The algorithm assigns to each vertex a "progress measure" from $M_G^\infty = M_G \cup \{\infty\}$, where

$$(20) \quad M_G = \begin{cases} [1]\times[n_1+1]\times[1]\times[n_3+1]\times\cdots[1]\times[n_{d-1}+1] & \text{if } d \text{ is even,}\\ [1]\times[n_1+1]\times[1]\times[n_3+1]\times\cdots[1]\times[n_{d-2}+1] & \text{if } d \text{ is odd.}\end{cases}$$

In other words, a progress measures is either $\infty$ or a length $d$ vector which at even index positions is 0, and at odd index positions $i$ ranges over $\{0,\dots,n_i\}$.

Initially, every vertex is assigned 0, the all-zero vector. The measures are repeatedly "incremented" in a certain fashion until a fixed point is reached.

We first explain the increment operation, which is at the heart of Jurdziński's algorithm. For $i < d$ and $x \in M_G^\infty$ we define $\langle x\rangle_i$ as follows. For $x = (l_0,\dots,l_{d-1})$, $\langle x\rangle_i = (l_0,\dots,l_i,0,0,\dots,0)$. In other words, we set positions indexed $> i$ to 0. Moreover, $\langle\infty\rangle_i = \infty$. We define a lexicographic total order on $M_G^\infty$, denoted $>$. Here, index 0 is the most significant position, and $\infty$ is greater than all other vectors. In addition, for $d$-vectors $x$ and $y$, we write $x >_i y$ iff $\langle x\rangle_i > \langle y\rangle_i$ according to the above ordering. For example, $(0,3,0,1) >_1 (0,2,0,3)$. Note that $x > y$ iff $x >_{d-1} y$. Now, we can say what it means to "increment" a progress measure. For each $i \in [d]$, let

$$(21) \qquad \mathrm{incr}_i(x) = \begin{cases} \langle x\rangle_i & \text{if } i \text{ is even } x \neq \infty,\\ \min\{y \in M_G^\infty \mid y >_i x\} & \text{if } i \text{ is odd, } x \neq \infty,\\ \infty & \text{if } x = \infty.\end{cases}$$

Observe that, for a fixed $i$, the operation $\mathrm{incr}_i(\cdot)$ is monotone with respect to the ordering $<$; that is, if $x \le x'$, then $\mathrm{incr}_i(x) \le \mathrm{incr}_i(x')$.

For simplicity in notation, if $v \in V$, we write $\langle x\rangle_v$ and $\mathrm{incr}_v(x)$ for $\langle x\rangle_{p(v)}$ and $\mathrm{incr}_{p(v)}(x)$, respectively. For every assignment $\rho\colon V \to M_G^\infty$ of progress measures to the vertices of a game graph, which we call an *assignment* for short, and for $v \in V$, let

$$(22) \qquad \text{best-nghb-ms}(\rho,v) = \begin{cases} \langle \min(\{\rho(w) \mid (v,w)\in E\})\rangle_v & \text{if } v \in V_0,\\ \langle \max(\{\rho(w) \mid (v,w)\in E\})\rangle_v & \text{if } v \in V_1.\end{cases}$$

Here, best-nghb-ms$(\rho,v)$ stands for the set of *best neighbors* of $v$ with respect to the *measure* we have defined. Jurdziński defines a "lifting" operator, which, given an assignment $\rho$ and $v \in V$, gives a new assignment. In order to define it, he first defines how an individual vertex's measure is "updated" with respect to those of its neighbors:

$$(23) \qquad \mathrm{update}(\rho,v) = \mathrm{incr}_v(\text{best-nghb-ms}(\rho,v)).$$

The "lifted" assignment, $\mathrm{lift}(\rho,v)\colon V \to D$, is then defined as follows:

$$(24) \qquad \mathrm{lift}(\rho,v)(u) = \begin{cases} \mathrm{update}(\rho,v) & \text{if } u = v,\\ \rho(u) & \text{otherwise.}\end{cases}$$

```
1 for $v \in V$ do
2   $\rho(v) := 0$
3 endfor
4 while there exists a $v$ such that update$(\rho, v) \neq \rho(v)$ do
5   $\rho := \text{lift}(\rho, v)$
6 endwhile
```

FIG. 1. *Jurdziński's lifting algorithm.*

Observe that for every $v$, the operator $\text{lift}(\cdot, v)$ is a monotone operator with respect to the complete partial ordering where $\rho \leq \rho'$ if $\rho(v) \leq \rho'(v)$ for all $v \in V$.

Jurdziński's algorithm is depicted in Figure 1. The outcome determines the winning set of vertices for each player as follows.

THEOREM 9 (see [17]). *Let $G$ be a parity game. Even has a winning strategy from precisely the vertices $v$ such that, after the lifting algorithm depicted in Figure* 1 *halts, $\rho(v) < \infty$.*

Jurdziński's algorithm needs at most $n' N_G$ iterations of the while loop where

$$N_G = |M_G^\infty| = 1 + \prod_{i:\ 0<2i+1\leq d-1} n_{2i+1}. \tag{25}$$

More precisely, Jurdziński argues as follows. Each vertex can only be lifted $N_G$ times. A lifting operation at $v$ can be performed in time $O(|\text{Sucs}(v)|)$, where $\text{Sucs}(v)$ denotes the set of successors of $v$. So, overall, he concludes, the running time is $O(m' N_G d)$. In this analysis, it is implicitly assumed that one can, in constant time, decide if there is a vertex $v$ such that update$(\rho, v) \neq \rho(v)$, and find such a vertex. We provide an implementation of Jurdziński's algorithm that achieves this.

Our algorithm, depicted in Figure 2, maintains a set $L$ of "pending" vertices $v$ whose measure needs to be considered for lifting, because a successor has recently been updated resulting in a requirement to update $\rho(v)$. This set $L$ is implemented as a list together with a bit array; extracting an element, adding an element, and membership test can then be carried out in constant time. Further, we maintain arrays $B$ and $C$ that store, for each vertex $v$, the value best-nghb-ms$(\rho, v)$ and the number of successors $u$ of $v$ with $\langle \rho(u) \rangle_v =$ best-nghb-ms$(\rho, v)$, denoted cnt$(\rho, v)$.

Whether a vertex $w$ needs to be placed on $L$ is determined in constant time by maintaining, for each vertex $w$, the current "best measure" $B(w)$ of any of its successors, as well as the count $C(w)$ of how many such neighbors there are with the "best measure." This is only necessary for $w \in V_0$, because if this is the case we need to be able to realize when all neighbors with the current minimum value have "died out," while for $w \in V_1$ we look at the maximum of all neighbors.

LEMMA 10. *The lifting algorithm depicted in Figure* 2 *computes the function $\rho$ from Jurdziński's algorithm in time $O(m' N_G d)$ and space $O(dm')$.*

*Proof.* The correctness follows from the above explanation. The running time follows because each vertex can enter $L$ at most $n_1 + 1$ times, and the time taken by the body of the while loop is proportional to the number of edges incident on the vertex.

The bound on the running time can be explained as follows. The initialization (lines 1–4) takes time $O(m'd)$. If a vertex enters $L$ in the body of the while loop, then its $\rho$-value will be incremented next time the vertex is removed from $L$. That means every vertex enters $L$ at most $N_G$ times. The time it takes to process a vertex $v$ taken

```
 1  foreach $v \in V$ do
 2    $B(v) := 0$; $C(v) := |\{w \mid (v,w) \in E\}|$; $\rho(v) := 0$;
 3  endfor
 4  $L := \{v \in V \mid p(v)$ is odd$\}$;
 5  while $L \neq \emptyset$ do
 6    let $v \in L$; $L := L \setminus \{v\}$;
 7    $t := \rho(v)$;
 8    $B(v) :=$ best-nghb-ms$(\rho, v)$; $C(v) :=$ cnt$(\rho, v)$; $\rho(v) := \mathrm{incr}_v(B(v))$;
 9    $P := \{w \in V \mid (w,v) \in E\}$;
10    foreach $w \in P$ such that $w \notin L$ do
11      if $w \in V_0$, $\langle t \rangle_w = B(w)$, and $\langle \rho(v) \rangle_w > B(w)$ then
12        if $C(w) = 1$ then $L := L \cup \{w\}$;
13        if $C(w) > 1$ then $C(w) := C(w) - 1$;
14      if $w \in V_1$ and $\langle \rho(v) \rangle_w > B(w)$ then
15        $L := L \cup \{w\}$;
16    endfor
17  endwhile
```

FIG. 2. *Efficient implementation of the lifting algorithm.*

from the loop is $O(\#$ vertices incident on $v)$. This means that lines 5–17 take time $O(m' N_G d)$. This proves the claim. ∎

We can now state one of our main theorems.

THEOREM 11. *For a Büchi automaton $A$, the relations $\preceq_f$, $\approx_f^{bi}$, and $\preceq_{de}$ can all be computed in time $O(|\Delta||Q|^3)$ and space $O(|Q||\Delta|)$.*

*Proof.* The theorem follows from Lemmas 4 and 10. Observe that in the (bi)simulation games involved we have $N_G = n_1 + 1 = O(|Q|^2)$. ∎

As mentioned, in prior work $O(|Q|^6)$–time and space [14], and $O(|Q|^{10})$–time and space [15] algorithms were given for deciding whether $q \preceq_f q'$, and, respectively, $q \approx_f^{bi} q'$, hold for a *single* pair of states $(q, q')$.

## 5. Reducing state spaces by quotienting: Delayed simulation is better.

In this section, we show that (1) quotienting with respect to delayed simulation preserves the recognized language; (2) this is not true with fair simulation; and (3) quotients with respect to delayed simulation can indeed be substantially smaller than quotients with respect to direct simulation, even when the latter is computed on the "accept closure" cl$(A)$ (unlike what we saw with delayed bisimulation). We first define quotients.

DEFINITION 12. For a Büchi automaton $A$, and an equivalence relation $\approx$ on the states of $A$, let $[q]$ denote the equivalence class of $q \in Q$ with respect to $\approx$. The *quotient* of $A$ with respect to $\approx$ is the automaton

$$A/\!\approx\; = \langle \Sigma, Q/\!\approx, \Delta_\approx, [q_I], F/\!\approx \rangle, \tag{26}$$

where

$$\Delta_\approx = \{([q], a, [q']) \mid \exists\, q_0 \in [q],\ q_0' \in [q'] \text{ such that } (q_0, a, q_0') \in \Delta\}. \tag{27}$$

In order to apply our simulation relations, we define, corresponding to each simulation preorder, an equivalence relation $\approx_o$, $\approx_{di}$, $\approx_{de}$, and $\approx_f$, where $q \approx_\star q'$ iff $q \preceq_\star q'$ and $q' \preceq_\star q$. Note that both $\approx_\star$ and $A/\!\approx_\star$ can be computed from $\preceq_\star$ requiring no more time (asymptotically) than that needed to compute $\preceq_\star$ on $A$. The quotient with respect to $\approx_{di}$ preserves the language of any automaton, while this is obviously not true for $\approx_o$. We will later see that this is not true for $\approx_f$ either. But first we show that this is true for $\approx_{de}$.

We start with a lemma.

LEMMA 13. *Let $A$ be a Büchi automaton.*

1. *If $q_0 \preceq_{de} q_0'$ and $(q_0, a, q_1) \in \Delta$, then there exists $q_1'$ with $q_1 \preceq_{de} q_1'$ and $(q_0', a, q_1') \in \Delta$.*

2. *If $q_0 \preceq_{de} q_0'$ and $[q_0]_{de}\, a_0\, [q_1]_{de}\, a_1 \dots$ is a finite or infinite run of $A/{\approx_{de}}$, then there exists a run $q_0' a_0 q_1' a_1 \dots$ of $A$ of the same length such that $q_i \preceq_{de} q_i'$ for every $i$.*

3. *If $q_0 \preceq_{de} q_0''$ and $[q_0]_{de}\, a_0\, [q_1]_{de}\, a_1 \dots$ is an infinite run of $A/{\approx_{de}}$ with $q_0 \in F$, then there exists a finite run $q_0'' a_0 \dots a_{r-1} q_r''$ of $A$ such that $q_j \preceq_{de} q_j''$ for $j \le r$ and $q_r'' \in F$.*

*Proof.* For the first part, recall that by definition of $\preceq_{de}$, we know Duplicator wins $G_A^{de}(q_0, q_0')$. Let $f$ be a winning strategy for him, and let $q_1' = f(q_0 q_0' a q_1)$. Then, by definition, $(q_0', a, q_1') \in \delta$. Also, it is easy to see that $f'$ defined by $f'(\rho) = f(q_0 q_0' a \rho)$ is a winning strategy for Duplicator in $G_A^{de}(q_1, q_1')$. Therefore, the claim holds.

For the second part, observe that by definition of $A/{\approx_{de}}$ there exist $\hat{q}_0$ and $\hat{q}_1$ such that (i) $q_0 \approx_{de} \hat{q}_0$, (ii) $q_1 \approx_{de} \hat{q}_1$, and (iii) $(\hat{q}_0, a_0, \hat{q}_1) \in \Delta$. From (i), we obtain $\hat{q}_0 \preceq_{de} q_0'$ by transitivity of $\preceq_{de}$. So, from (iii) and the first part of the lemma, we can conclude there exists $(q_0', a, q_1') \in \Delta$ such that $\hat{q}_1 \preceq_{de} q_1'$. From (ii), we obtain $q_1 \preceq_{de} q_1'$. Hence, we have constructed the first transition of the desired run and are in a completely analogous situation. The rest follows by induction.

For the third part, first set $q_0' = q_0$. Let $q_0' a_0 q_1' a_1 \dots$ be the infinite run which we know exists by the second part. Next, let $f$ be a winning strategy of Duplicator in $G_A^{de}(q_0', q_0'')$. Consider $q_0'' a_0 q_1'' a_1 \dots$ defined by $q_{i+1}'' = f(q_0' q_0'' a_0 \dots q_i')$. Just as in the proof of the first part, it can be argued that $q_j \preceq_{de} q_j' \preceq_{de} q_j''$ holds for every $j$. Because of $q_0' = q_0 \in F$, we conclude there exists $r$ such that $q_r'' \in F$, which completes the proof. ∎

THEOREM 14. *For any Büchi automaton $A$, $L(A) = L(A/{\approx_{de}})$.*

*Proof.* To see that $L(A) \subseteq L(A/{\approx_{de}})$, consider any accepting run $\pi = q_0 a_0 q_1 a_1 \dots$ of $A$. By definition of $A/{\approx_{de}}$, $[q_0]a_0[q_1]a_1 \dots$ is an accepting run of $A/{\approx_{de}}$. This holds for any of our simulation notions.

To show $L(A/{\approx_{de}}) \subseteq L(A)$, consider an accepting run $[q_0]a_0[q_1]a_1 \dots$ of $A/{\approx_{de}}$. Although we cannot guarantee that $q_0 a_0 q_1 a_1 \dots$ is a run of $A$, we can construct another accepting run over the same word.

We may assume that $q_0 = q_I$ and that there are infinitely many $i$ such that $q_i \in F$. We construct a sequence $\rho_0$, $\rho_1$, ... of finite runs of $A$ on prefixes of $a_0 a_1 \dots$ where $\rho_{l+1}$ strictly extends $\rho_l$ and contains at least $l+1$ elements from $F$. So the limit of the $\rho_i$'s will be the run we are looking for.

We start with $\rho_0 = q_0$. Assume $\rho_l = q_0' a_0 \dots q_i'$ has already been constructed in such a way that $q_i \preceq_{de} q_i'$. There exists $j > i$ such that $q_j \in F$. So, by the second part of the previous lemma, we know there exists a run $q_i' a_i \dots q_j'$ such that $q_j \preceq_{de} q_j'$. By the third part of the lemma, we know there exists $k \ge j$ and a run $q_j' a_j \dots q_k'$ such that $q_k \preceq_{de} q_k'$ and $q_k' \in F$. We set $\rho_{l+1} = \rho a_i q_{i+1}' \dots q_k'$. ∎

We can thus use $A/{\approx_{de}}$ to reduce the size of $A$, just as with direct simulation. In fact, $A/{\approx_{de}}$ can be smaller than $A/{\approx_{di}}$ (as well as $\mathrm{cl}(A)/{\approx_{di}}$) by an arbitrarily large factor.

PROPOSITION 15. *For $n \ge 2$, there is a Büchi automaton $A_n$ with $n+1$ states*

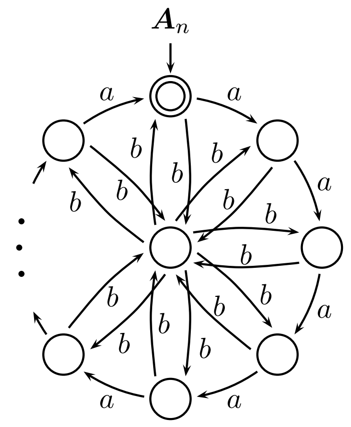


FIG. 3. *Family of automata $A_n$.*

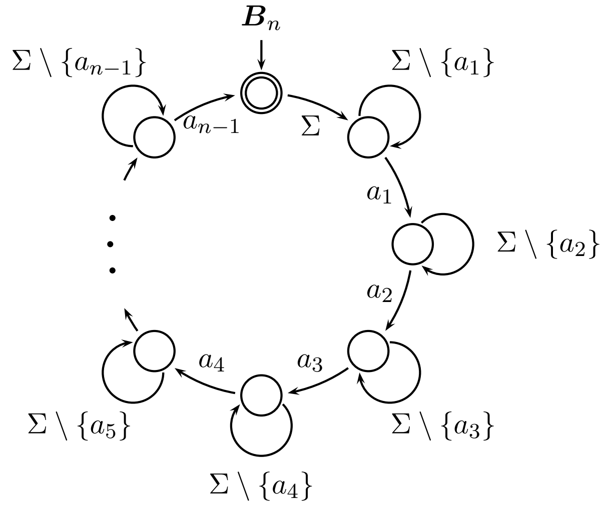

$\boldsymbol{B}_n$
$\Sigma \setminus \{a_{n-1}\}$
$\Sigma \setminus \{a_1\}$
$a_{n-1}$
$\Sigma$
$a_1$
$\Sigma \setminus \{a_2\}$
$a_2$
$a_3$
$a_4$
$\Sigma \setminus \{a_3\}$
$\Sigma \setminus \{a_4\}$
$\Sigma \setminus \{a_5\}$

FIG. 4. *Family of automata $B_n$.*

*such that $A_n/{\approx_{de}}$ has 2 states but $A_n/{\approx_{di}}$ has $n + 1$ states (and $A_n = \mathrm{cl}(A_n)$).*

*Proof.* Consider automaton $A_n$ in Figure 3. It is not hard to establish that in $A_n$ each outer state delayed simulates each other outer state. Thus $A_n/{\approx_{de}}$ has 2 states. On the other hand, $A_n = \mathrm{cl}(A_n)$, and no state of $A_n$ direct simulates any other state of $A_n$. Thus $A_n/{\approx_{di}} = A_n$ and has $n + 1$ states. ▯

Next we see that Theorem 14 fails badly for fair simulation and bisimulation; that is, fair (bi)simulation cannot be used for state space reduction via quotienting under any reasonable definition of quotient. [15] already makes a very closely related observation, showing an automaton whose fair bisimulation quotient is not fair bisimilar to itself.

PROPOSITION 16. *For $n \geq 2$, there is a Büchi automaton $B_n$ with $n$ states, each of which fairly (bi)simulates every other state, but such that no Büchi automaton with fewer than $n$ states accepts $L(B_n)$. In particular, $L(B_n) \neq L(B_n/{\approx_f^{bi}})$.*

*Proof.* Consider the automaton $B_n$ shown in Figure 4. It has $n$ states and an alphabet $\Sigma = \{a_1, \ldots, a_{n-1}\}$. To see that every state of $B_n$ fair simulates (and fair bisimulates) every other state, first note that because the automaton is deterministic Duplicator has no choice in her strategy. A run (played by Spoiler) goes through the accept state infinitely often iff each $a_i$ is encountered infinitely often. But this statement holds no matter which state the run begins from. Thus Duplicator's unique strategy from the initial state pair will be a winning strategy. The language $L(B_n)$ contains precisely those $\omega$-words where each $a_i$ occurs infinitely often. It is not hard to show that there are no Büchi automata recognizing $L(B_n)$ with fewer than $n$ states. $\square$

**6. Conclusions.** We have presented a unified parity-game framework in which to understand optimal known algorithms for a variety of simulation notions for Büchi automata. In particular, we have improved upon the best bounds for fair simulation (and fair bisimulation), matched the best bound for ordinary simulation, and presented an algorithm for the new notion of delayed simulation. Our algorithms employ a relatively simple fixed point computation, an implementation of an algorithm by Jurdziński for parity games, and should perform well in practice.

Our own main aim in using simulations is efficient state space reduction, as in [8]. We introduced delayed simulation and showed that, unlike fair simulation, delayed simulation quotients can be used for state space reduction, and allow greater reduction than direct (strong) simulation, which has been used in the past. Optimization of property automata prior to model checking is an ingredient in making explicit state model checkers such as SPIN more efficient. Preliminary results indicate that in practice delayed simulation does outperform direct simulation on many inputs; further studies need to be carried out to get a clearer picture of the relative advantages of delayed simulation.

## REFERENCES

[1] H. R. Andersen, *Model checking and Boolean graphs*, Theoret. Comput. Sci., 126 (1994), pp. 3–30.

[2] B. Bloom and R. Paige, *Transformational design and implementation of a new efficient solution to the ready simulation problem*, Sci. Comput. Programming, 24 (1995), pp. 189–220.

[3] D. Bustan and O. Grumberg, *Checking for Fair Simulation in Models with Büchi Fairness Constraints*, Tech. report TR-CS-2000-13, Technion, Haifa, Israel, 2000.

[4] D. Bustan and O. Grumberg, *Applicability of fair simulation*, Inform. and Comput., 194 (2004), pp. 1–18.

[5] D. Bustan and O. Grumberg, *Simulation-based minimization*, ACM Trans. Comput. Logic, 4 (2003), pp. 181–206.

[6] D. L. Dill, A. J. Hu, and H. Wong-Toi, *Checking for language inclusion using simulation preorders*, in Proceedings of the 3rd International Workshop on Computer Aided Verification, CAV '91, Aalborg, Denmark, 1991, Lecture Notes in Comput. Sci. 575, K. G. Larsen and A. Skou, eds., Springer-Verlag, Berlin, 1992, pp. 255–265.

[7] K. Etessami, *A hierarchy of polynomial-time computable simulations for automata*, in Proceedings of the 13th International Conference of Concurrency Theory, CONCUR 2002, Brno, Czech Republic, 2002, Lecture Notes in Comput. Sci. 2421, L. Brim, P. Jancar, M. Kretínský, and A. Kucera, eds., Springer-Verlag, Berlin, 2002, pp. 131–144.

[8] K. Etessami and G. J. Holzmann, *Optimizing Büchi automata*, in Proceedings of the 11th International Conference of Concurrency Theory, CONCUR 2000, University Park, PA, 2000, Lecture Notes in Comput. Sci. 1877, C. Palamidessi, ed., Springer-Verlag, Berlin, 2000, pp. 153–167.

[9] K. Etessami, R. Schuller, and Th. Wilke, *Fair simulation relations, parity games, and state space reduction for Büchi automata*, in Proceedings of the 28th International Colloquium on Automata, Languages and Programming, ICALP 2001, Crete, Greece, 2001, Lecture Notes in Comput. Sci. 2076, F. Orejas, P. G. Spirakis, and J. van Leeuwen, eds., Springer-Verlag, Berlin, 2001, pp. 694–707.

[10] C. Fritz and Th. Wilke, *State space reductions for alternating Büchi automata: Quotienting by simulation equivalences*, in Proceedings of the 22nd Conference on Foundations of Software Technology and Theoretical Computer Science, FST TCS 2002, Kanpur, India, 2002, Lecture Notes in Comput. Sci. 2556, M. Agrawal and A. Seth, eds., Springer-Verlag, Berlin, pp. 157–169.

[11] O. Grumberg and D. Long, *Model checking and modular verification*, ACM Trans. Programming Languages and Systems, 16 (1994), pp. 843–871.

[12] E. Grädel, W. Thomas, and Th. Wilke, eds., *Automata, Logics, and Infinite Games: A Guide to Current Research*, Lecture Notes in Comput. Sci. 2500, Springer-Verlag, New York, 2002.

[13] S. Gurumurthy, R. Bloem, and F. Somenzi, *Fair simulation minimization*, in Proceedings of the 14th International Conference on Computer Aided Verification, CAV 2002, Copenhagen, Denmark, 2002, Lecture Notes in Comput. Sci. 2404, E. Brinksma and K. Guldstrand Larsen, eds., Springer-Verlag, Berlin, 2002, pp. 610–624.

[14] T. A. Henzinger, O. Kupferman, and S. Rajamani, *Fair simulation*, Inform. and Comput., 173 (2002), pp. 64–81.

[15] T. A. Henzinger and S. Rajamani, *Fair bisimulation*, in Proceedings of the 6th International Conference on Tools and Algorithms for Construction and Analysis of Systems, TACAS 2000, Berlin, Germany, 2000, Lecture Notes in Comput. Sci. 1785, S. Graf and M. I. Schwartzbach, eds., Springer-Verlag, Berlin, 2000, pp. 299–314.

[16] M. Henzinger Rauch, T. A. Henzinger, and P. W. Kopke, *Computing simulations on finite and infinite graphs*, in Proceedings of the 36th Annual Symposium on Foundations of Computer Science, Milwaukee, WI, 1995, IEEE, pp. 453–462.

[17] M. Jurdziński, *Small progress measures for solving parity games*, in Proceedings of the 17th Annual Symposium on Theoretical Aspects of Computer Science STACS 2000, Lille, France, 2000, Lecture Notes in Comput. Sci. 1770, H. Reichel and S. Tison, eds., Springer-Verlag, Berlin, 2000, pp. 290–301.

[18] N. Klarlund, *Progress measures, immediate determinacy, and a subset construction for tree automata*, Ann. Pure Appl. Logic, 69 (1994), pp. 243–268.

[19] O. Kupferman and M. Y. Vardi, *Weak alternating automata and tree automata emptiness*, in Proceedings of the 30th Annual ACM Symposium on the Theory of Computing, Dallas, TX, 1998, ACM, New York, pp. 224–233.

[20] R. Milner, *Communication and Concurrency*, Prentice-Hall, New York, 1989.

[21] R. Paige and R. E. Tarjan, *Three partition refinement algorithms*, SIAM J. Comput., 16 (1987), pp. 973–989.

[22] F. Somenzi and R. Bloem, *Efficient Büchi automata from LTL formulae*, in Proceedings of the 12th International Conference on Computer Aided Verification, CAV 2000, Chicago, IL, 2000, Lecture Notes in Comput. Sci. 1855, E. A. Emerson and A. Prasad Sistla, eds., Springer-Verlag, Berlin, 2000, pp. 248–263.

[23] W. Thomas, *Automata on infinite objects*, in Handbook of Theoretical Computer Science, Vol. B: Formal Methods and Semantics, J. van Leeuwen, ed., Elsevier, Amsterdam, 1990, pp. 134–191.

[24] W. Thomas, *Languages, automata and logic*, in Handbook of Formal Languages, Vol. 3: Beyond Words, A. Salomaa and G. Rozenberg, eds., Springer-Verlag, Berlin, 1997, pp. 389–455.

[25] I. Walukiewicz, *Completeness of Kozen's axiomatisation of the propositional $\mu$-calculus*, Inform. and Comput., 157 (2000), pp. 142–182.

SIAM J. COMPUT. © 2005 Society for Industrial and Applied Mathematics
Vol. 34, No. 5, pp. 1176–1195

# THE COMPLEXITY OF FINDING PATHS IN GRAPHS WITH BOUNDED INDEPENDENCE NUMBER[*]

ARFST NICKELSEN[†] AND TILL TANTAU[†]

**Abstract.** We study the problem of finding a path between two vertices in finite directed graphs whose independence number is bounded by some constant $k$. The independence number of a graph is the largest number of vertices that can be picked such that there is no edge between any two of them. The complexity of this problem depends on the exact question we ask: Do we wish only to tell whether a path exists? Do we also wish to construct such a path? Are we required to construct the shortest one? Concerning the first question, we show that the reachability problem is first-order definable for all $k$ and that its succinct version is $\Pi_2^{\mathrm{P}}$-complete for all $k$. In contrast, the reachability problems for many other types of finite graphs, including dags and trees, are not first-order definable, and their succinct versions are PSPACE-complete. Concerning the second question, we show not only that we can construct paths in logarithmic space, but that there even exists a logspace approximation scheme for this problem. The scheme gets a ratio $r > 1$ as additional input and outputs a path that is at most $r$ times as long as the shortest path. Concerning the third question, we show that even telling whether the shortest path has a certain length is NL-complete and thus is as difficult as for arbitrary directed graphs.

**Key words.** reachability, connectivity, shortest paths, distance in graphs, logarithmic space, tournaments, first-order definability, polynomial hierarchy, completeness, approximation algorithms, succinct representations

**AMS subject classifications.** 03C13, 05C12, 05C20, 05C62, 05C69, 05C85, 68Q17, 68Q19, 68R10, 68W25

**DOI.** 10.1137/S0097539704441642

## 1. Introduction.

Finding paths in graphs is one of the most fundamental problems in graph theory and has both practical and theoretical applications in many different areas. For this problem we are given a graph $G$ and two vertices $s$ and $t$, the *source* and the *target*, and we are asked to find a path from $s$ to $t$. The problem comes in different versions: The most basic is the *reachability problem*, which just asks whether such a path *exists*. This problem is also known as the "graph accessibility problem" or "s-t-connectivity problem." The *construction problem* asks us to *construct* a path, provided one exists. The *optimization problem* asks us to construct not just any path, but a *shortest* one. Closely related to the optimization problem is the *distance problem*, which asks us to decide whether the distance of $s$ and $t$ is bounded by a given number. If the optimization problem is difficult to solve, we can consider the *approximation problem*, which asks us to construct a path that is not necessarily a shortest path, but that is no longer than the distance of $s$ and $t$ times a constant factor.

In this paper we show that for directed graphs whose independence number is bounded by some constant $k$, the reachability problem, the construction problem,

[*]Received by the editors March 3, 2004; accepted for publication (in revised form) December 29, 2004; published electronically June 16, 2005. The authors presented parts of this paper at the 8th Computing and Combinatorics Conference, 2002 [22], and the 21st Symposium on Theoretical Aspects of Computer Science, 2004 [29].

http://www.siam.org/journals/sicomp/34-5/44164.html

[†]Fakultät für Elektrotechnik und Informatik, Technische Universität Berlin, 10623 Berlin, Germany (nicke@cs.tu-berlin.de, tantau@cs.tu-berlin.de). The second author was supported in part by an Erwin-Stephan-Prize grant of the Technical University of Berlin and a post-doc research fellowship grant of the German Academic Exchange Service (DAAD). The second author's work was done in part at the University of Rochester and at the International Computer Science Institute in Berkeley.

and the optimization problem have fundamentally different computational complexities. The *independence number* $\alpha(G)$ of an (undirected or directed) graph $G$ is the maximum number of vertices that can be picked from $G$ such that there is no edge between any two of these vertices. Our main results are the following:

1. The reachability problem for finite graphs with independence number at most $k$ is first-order definable for all $k$. This does not hold for infinite graphs for any $k$.
2. The reachability problem for succinctly represented graphs with independence number at most $k$ is $\Pi_2^{\mathrm{P}}$-complete for all $k$.
3. There exists an algorithm that needs space $O\big((\alpha(G) + \log m) \log n\big)$ on $n$-vertex graphs to compute a path between two given connected vertices whose length is at most $1 + 1/m$ times their distance.
4. The distance problem for graphs with independence number at most $k$ is NL-complete for all $k$.

The most prominent examples of graphs with bounded independence number are *tournaments* [21, 24], which are directed graphs with exactly one edge between any two vertices. Their independence number is 1. The reachability problem for tournaments arises naturally if we try to rank or sort objects according to a comparison relation that for any two objects tells us which one "beats" the other, but which is not necessarily acyclic. A different example of graphs with bounded independence number, studied in [7], are directed graphs $G = (V, E)$ whose underlying undirected graph is claw-free, i.e., does not contain the graph $K_{1,m}$ for some constant $m$, and whose minimum degree is at least $|V|/3$. Their independence number is at most $3m - 3$.

To gain some understanding of the behavior of the independence number function, first note that independence is a monotone graph property: adding edges to a graph can only decrease the independence number, and deleting edges can only increase it. Given two graphs with the same vertex set and independence numbers $\alpha$ and $\alpha'$, the independence number of their union is at most the minimum of $\alpha$ and $\alpha'$, and the independence number of their disjoint union is $\alpha + \alpha'$. Thus if a graph consists of, say, four disjoint tournaments with arbitrary additional edges connecting these tournaments, its independence number is at most 4. Intuitively, a graph with a low independence number must have numerous edges, and, indeed, at least $\binom{n}{2} \,/\, \binom{\alpha(G)+1}{2}$ edges must be present in any $n$-vertex graph $G$. This abundance of edges might suggest that if paths between two given vertices exist, there should also exist a short path between them. While this is true for the undirected case, in the directed case (which interests us in this paper) the distance between two vertices can become as large as $n - 1$, even in $n$-vertex tournaments.

**1.1. How difficult is it to tell whether a path exists?** The reachability problem for finite directed graphs, which will be denoted REACH in the following, is well known to be NL-complete [16, 17] and thus "easy" from a computational point of view. The complexity of the reachability problem drops if we restrict the type of graphs for which we try to solve it. The reachability problem $\text{REACH}_{\text{u}}$ for finite undirected graphs is SL-complete [19] and thus presumably easier to solve. The even more restricted problem $\text{REACH}_{\text{forest}}$ for directed forests and the problem $\text{REACH}_{\text{out}\le 1}$ for directed graphs in which all vertices have out-degree at most 1 are L-complete [4]. Here and in the following, "completeness" is meant with respect to first-order reductions ($\le_{\text{fo}}$-reductions) in the sense of Immerman [15], where ordering and the bit predicate are available. Barrington, Immerman, and Straubing [2] have shown that first-order reductions are the same as DLOGTIME-uniform many-one $\mathrm{AC}^0$-reductions.

The complexity of the reachability problem for finite directed graphs whose independence number is bounded by a constant $k$ is much lower: Somewhat surprisingly, this problem is first-order definable for all $k$. Formally, for each $k$ the language $\text{REACH}_{\alpha \leq k} := \text{REACH} \cap \{\langle G, s, t\rangle \mid \alpha(G) \leq k\}$ is first-order definable, where $\langle\,\rangle$ denotes a standard binary encoding. First-order definability means the following: Let $\tau = (\mathrm{E}^2, \mathrm{s}, \mathrm{t})$ be the signature of directed graphs with two distinguished vertices. The binary relation symbol E represents an edge relation, and the constant symbols s and t represent a source and a target vertex. We show that for each $k$ there exists a first-order formula $\phi_{\text{reach},\alpha\leq k}$ over the signature $\tau$ for which the following holds: For all *finite* directed graphs $G = (V, E)$ and all $s, t \in V$ the $\tau$-structure $(V, E, s, t)$ is a model of $\phi_{\text{reach},\alpha\leq k}$ if

1. $\alpha(G) \leq k$ and
2. there is path from $s$ to $t$ in $G$.

The formulas will require neither an ordering on the universe nor the bit predicate; see [15] for a discussion of the importance of these predicates. Note, however, that we implicitly use these predicates in the first-order reductions that we present.

Languages whose descriptive complexity is first-order are known to be very simple from a computational point of view. They can be decided by a family of DLOGTIME-uniform $\mathrm{AC}^0$-circuits [2], in constant parallel time on concurrent-read, concurrent-write parallel random access machines [14], and they constitute a small subclass of the class L of logarithmic space. Since it is known that L-hard sets cannot be first-order definable [1, 9], $\text{REACH}_{\alpha\leq k}$ is *unconditionally* easier to solve than REACH, $\text{REACH}_{\text{u}}$, and $\text{REACH}_{\text{forest}}$. It is even easier than the seemingly trivial reachability problem for graphs that are directed paths, which Etessami [6] has shown to be L-complete. For the special case of tournaments, conditions for strong connectedness (and thus, implicitly, for reachability) were proved in [12], but these conditions yield weaker bounds on the complexity of the reachability problem for tournaments than those shown in the present paper.

Our results on the first-order definability of $\text{REACH}_{\alpha\leq k}$ apply only to finite graphs. Let $\text{REACH}^{\infty}_{\alpha\leq k}$ denote the class of all triples $(G, s, t)$ such that $G$ is a (possibly infinite) directed graph with $\alpha(G) \leq k$ in which there is a path from $s$ to $t$. We show that no set of first-order formulas (not even an infinite one) has $\text{REACH}^{\infty}_{\alpha\leq k}$ as its class of models for any $k$.

When studying the complexity of a graph problem, one usually assumes (as we did above) that the input graph is encoded as a binary string "in some standardized way," for example, using adjacency lists. Which particular encoding method is chosen is of little or no concern for the computational complexity of the problem. This is no longer true if the input graphs are encoded *succinctly*, as is often the case, for instance, in hardware design. Succinctly represented graphs are given indirectly via a program or a circuit that decides the edge relation of the graph. For reachability problems, Papadimitriou and Yannakakis [23] and Wagner [35, 36] have shown that SUCCINCT-REACH, $\text{SUCCINCT-REACH}_{\text{u}}$, $\text{SUCCINCT-REACH}_{\text{forest}}$, and $\text{SUCCINCT-REACH}_{\text{out}\leq 1}$ are all PSPACE-complete. In contrast to this, we show that $\text{SUCCINCT-REACH}_{\alpha\leq k}$ is complete for the second level of the polynomial hierarchy, more precisely for $\Pi^{\mathrm{P}}_2$, for all $k$. Interestingly, the succinct version of the dominating set problem for tournaments is complete for $\Sigma^{\mathrm{P}}_2$, as shown by Umans [33].

**1.2. How difficult is it to construct a path?** The low complexity of the reachability problem seemingly settles the complexity of finding paths in graphs with bounded independence number. At first sight, the path construction problem appears to reduce to the reachability problem via a simple algorithm: Starting at the source vertex, for each successor of the current vertex check whether we can reach the target from it (for at least one successor this test will be true); make that successor the current vertex; and repeat until we have reached the target. Unfortunately, this algorithm is flawed since it can lead us around in endless cycles in graphs that are not acyclic. A correct algorithm does not move to an *arbitrary* successor, but to the successor that is *nearest* to the target. This corrected algorithm produces not only *some* path, but a shortest one. However, the algorithm now needs to compute the distance between vertices internally, which is conceptually a more difficult problem than deciding whether two vertices are connected.

Nevertheless, we show that a path between any two connected vertices can be constructed in logarithmic space in graphs with bounded independence number. There even exists a *logspace approximation scheme* for this problem. This means that for each $r > 1$ and each $k$ there exists a logspace-computable function that maps an input $\langle G, s, t\rangle$ with $\alpha(G) \leq k$ to a path from $s$ to $t$ of length at most $r$ times the distance of $s$ and $t$. If no path exists, the function outputs "no path exists."

**1.3. How difficult is it to construct a shortest path?** Our final result settles the complexity of constructing a *shortest* path in a graph with bounded independence number. We show that even for tournaments this problem is as difficult as constructing a shortest path in an arbitrary graph. As pointed out above, the complexity of constructing a shortest path hinges on the complexity of the *distance problem* $\text{DISTANCE}_{\text{tourn}} := \big\{\langle G, s, t, d\rangle \mid G \text{ is a tournament in which there is a path from } s \text{ to } t \text{ of length at most } d\big\}$. We prove that this problem is NL-complete. Thus DISTANCE and $\text{DISTANCE}_{\text{tourn}}$ are $\leq_{\text{fo}}$-equivalent, but REACH and $\text{REACH}_{\text{tourn}}$ are not. The succinct version of $\text{DISTANCE}_{\text{tourn}}$ is PSPACE-complete.

**1.4. Organization of this paper.** In section 2 we study graph-theoretic definitions and results. We prove a general theorem that relates the independence number of a graph to its different domination numbers. We believe this theorem to be of independent interest. In section 3 we study the reachability problem. We show that the problem $\text{REACH}_{\alpha\leq k}$ is first-order definable by explicitly giving a defining formula, that the infinite version is not first-order definable, and that the succinct version is $\Pi_2^{\text{P}}$-complete. In section 4 we present a logspace approximation scheme for constructing paths in graphs with bounded independence number. In section 5 we prove that the distance problem for tournaments is NL-complete and that its succinct version is PSPACE-complete.

## 2. Graph-theoretic definitions and results.

In this section we first fix the notation and terminology for basic graph-theoretic concepts. Then we prove a generalization of the so-called Lion King lemma; see Theorem 2.2. At the end of the section we prove Theorem 2.3, which will be the crucial building block of our first-order definition of $\text{REACH}_{\alpha\leq k}$.

A *directed graph* or just a *graph* is a nonempty set $V$ of *vertices* together with a set $E \subseteq V \times V$ of directed *edges*. A *tournament* is a graph with exactly one edge between any two different vertices and $(v, v) \notin E$ for all $v \in V$. A (rooted) *tree* is a graph in which there is a unique path from the root vertex to each vertex. A *forest* is the disjoint union of trees.

The *out-degree* of a vertex $u$ is the number of vertices $v$ with $(u, v) \in E$. A *path of length* $\ell$ in a graph $G = (V, E)$ is a sequence $(v_1, \dots, v_{\ell+1})$ of distinct vertices with $(v_i, v_{i+1}) \in E$ for $i \in \{1, \dots, \ell\}$. A vertex $t$ is *reachable* from a vertex $s$ if there is a path from $s$ to $t$. The *distance* $\mathrm{d}(s,t)$ of two vertices is the length of the shortest path between them or is undefined, if no path exists. For sets $U, U' \subseteq V$ let $\mathrm{d}(U,U') := \min\{\mathrm{d}(u,u') \mid u \in U, u' \in U'\}$. For $i \in \mathbb{N}$, a vertex $u \in V$ is said to *$i$-dominate* a vertex $v \in V$ if $\mathrm{d}(u,v) \le i$. Let $\mathrm{dom}_i(U)$ denote the set of all vertices that are $i$-dominated by vertices in $U$. A set $U \subseteq V$ is an *$i$-dominating set* for $G$ if $\mathrm{dom}_i(U) = V$. The *$i$-domination number* $\beta_i(G)$ is the minimum size of an $i$-dominating set for $G$. A set $U \subseteq V$ is *independent* if there is no edge in $E$ connecting vertices in $U$. The maximum size of independent sets in $G$ is its *independence number* $\alpha(G)$.

Turán [31], referenced in [32], gives an exact formula for the minimum number of edges in a graph as a function of the independence number. However, the simpler bound from the following lemma will be more appropriate for our purposes.

Lemma 2.1. *Let $G = (V,E)$ be a finite graph, $n := |V|$, $\alpha := \alpha(G) < n$. Then $G$ has at least $\binom{n}{2} \,/\, \binom{\alpha+1}{2}$ edges, and there exists a vertex with out-degree at least $(n-1) \,/\, 2\binom{\alpha+1}{2}$.*

*Proof.* The number of $(\alpha+1)$-element subsets of $V$ is $\binom{n}{\alpha+1}$ for $\alpha < n$. Every such set contains two vertices linked by an edge. Every such edge is in $\binom{n-2}{\alpha-1}$ different $(\alpha+1)$-element subsets of $V$. Therefore there are at least $\binom{n}{\alpha+1} \,/\, \binom{n-2}{\alpha-1} = \binom{n}{2} \,/\, \binom{\alpha+1}{2}$ edges in $G$. The average out-degree in $G$ hence must be at least

$$\frac{1}{n}\frac{\binom{n}{2}}{\binom{\alpha+1}{2}} = \frac{n-1}{2\binom{\alpha+1}{2}}.$$

Since this is a lower bound on the *average* out-degree, at least one vertex must have at least this out-degree. ∎

Theorem 2.2. *Let $G = (V,E)$ be a finite graph with at least two vertices, $n := |V|$, $\alpha := \alpha(G) < n$, and $c := (\alpha^2+\alpha)/(\alpha^2+\alpha-1)$. Then*

1. *$\beta_1(G) \le \lceil \log_c n \rceil$ and*
2. *$\beta_2(G) \le \alpha$.*

*Proof.* We greedily construct a 1-dominating set $D_1$ for $G$ of size at most $\lceil \log_c n \rceil$. In each step we put a vertex $v_i$ into $D_1$ that dominates as many vertices as possible of the subset $V_i \subseteq V$ not dominated so far. Formally, let $V_0 := V$ and for $i \ge 0$, as long as $V_i$ is not empty, choose a vertex $v_i \in V_i$ such that $V_{i+1} := V_i \setminus \mathrm{dom}_1\bigl(\{v_i\}\bigr)$ is as small as possible. Let $i_{\max}$ be the first $i$ such that $V_i$ is empty. By Lemma 2.1 the out-degree of each $v_i$ with $i \in \{0, \dots, i_{\max}-1\}$ is at least $\bigl(|V_i|-1\bigr) \,/\, 2\binom{\alpha+1}{2}$ and thus

$$\begin{aligned} |V_{i+1}| &\le |V_i| - 1 - \frac{|V_i|-1}{2\binom{\alpha+1}{2}} < |V_i| - \frac{|V_i|}{2\binom{\alpha+1}{2}} \\ &= |V_i|\left(1 - \frac{1}{2\binom{\alpha+1}{2}}\right) = |V_i|\left(\frac{\alpha^2+\alpha-1}{\alpha^2+\alpha}\right) = \frac{|V_i|}{c}. \end{aligned}$$

This shows that the size of $V_i$ decreases by at least the factor $c$ in each step. Thus after at most $\lceil \log_c n \rceil$ iterations the set $V_i$ is empty and $D_1 := \{v_0, \dots, v_{i_{\max}-1}\}$ is the desired 1-dominating set.

We next construct a 2-dominating set $D_2$ of size at most $\alpha$ by removing superfluous vertices from $D_1$. Formally, let $W_{i_{\max}} := \emptyset$ and

$$W_i := \begin{cases} W_{i+1} & \text{if } v_i \in \mathrm{dom}_1(W_{i+1}), \\ W_{i+1} \cup \{v_i\} & \text{otherwise.} \end{cases}$$

Clearly, $D_2 := W_0$ is a 2-dominating set. To prove $|D_2| \leq \alpha$, assume that $D_2$ contains at least $\alpha + 1$ vertices $v_{i_1}, \dots, v_{i_{\alpha+1}} \in D_1$. Since the set of these vertices cannot be independent, there must exist indices $i_r$ and $i_s$ such that $(v_{i_r}, v_{i_s}) \in E$. By construction of the set $D_1$, this can be the case only if $i_s > i_r$. But then $v_{i_r} \notin D_2$ by construction of $W_{i_r}$, a contradiction. ∎

For tournaments $G$, Theorem 2.2 yields $\beta_1(G) \leq \lceil \log_2 n \rceil$ and $\beta_2(G) = 1$. The first result was first proved by Megiddo and Vishkin in [20], where it was used to show that the dominating set problem for tournaments is not NP-complete, unless $\mathrm{NP} \subseteq \mathrm{DTIME}\bigl[n^{O(\log n)}\bigr]$. The second result is also known as the Lion King lemma, which was first noticed by Landau [18] in the study of animal societies, where the dominance relations in prides of lions form tournaments. It has applications in the study of P-selective sets [13] and many other fields.

THEOREM 2.3. *Let $G = (V, E)$ be a finite graph with at least two vertices, $n := |V|$, $\alpha := \alpha(G) < n$, $c := (\alpha^2 + \alpha)/(\alpha^2 + \alpha - 1)$, and $s, t \in V$. Then the following statements are equivalent:*

1. *There is no path from $s$ to $t$ in $G$.*
2. *There is a subset $D_1 \subseteq V$ with $|D_1| \leq \lceil \log_c n \rceil$ such that $\mathrm{dom}_1(D_1)$ is closed under reachability, $s \in \mathrm{dom}_1(D_1)$ and $t \notin \mathrm{dom}_1(D_1)$.*
3. *There is a subset $D_2 \subseteq V$ with $|D_2| \leq \alpha$ such that $\mathrm{dom}_2(D_2)$ is closed under reachability, $s \in \mathrm{dom}_2(D_2)$ and $t \notin \mathrm{dom}_2(D_2)$.*

*Proof.* The second and third statements imply the first since no path starting at a vertex $s$ inside a set that is closed under reachability can "leave" this set to arrive at a vertex $t$ outside this set. To show that the first statement implies the second, consider the set $S$ of vertices reachable from $s$ in $G$. Then $S$ is closed under reachability, $s \in S$ and $t \notin S$. The induced graph $G' := \bigl(S, E \cap (S \times S)\bigr)$ also has independence number at most $\alpha$. If $G'$ contains at least two vertices, Theorem 2.2 tells us that the graph $G'$ has a 1-dominating set $D_1$ of size at most $\lceil \log_c n \rceil$. If $G'$ contains only one vertex, it trivially has a 1-dominating set of size $1 \leq \lceil \log_c n \rceil$. To show that the first statement implies the third, consider the same graph $G'$ once more. By Theorem 2.2 it also has a 2-dominating set $D_2$ of size at most $\alpha$. ∎

## 3. Complexity of the reachability problem.

In this section we answer the following question: How difficult is it to tell whether there is a path between two vertices in a directed graph with bounded independence number? In the first subsection, we answer it for finite graphs, in the second for infinite ones, and in the third for succinctly represented graphs.

### 3.1. First-order definability of the reachability problem.

The reachability problem for graphs with bounded independence number is first-order definable. Before we prove this claim, let us review some basic notions from descriptive complexity theory.

*First-order definability* is a language property. It can be defined as follows for the special case of languages $A \subseteq \{\langle V, E, s, t\rangle \mid (V, E) \text{ is a finite graph}, s, t \in V\}$: Let $\tau = (\mathrm{E}^2, \mathrm{s}, \mathrm{t})$ be the *signature of graphs with two designated vertices.* A *first-order $\tau$-formula* is a first-order formula that contains, other than quantifiers, variables, and connectives, only the binary relation symbol E and the constant symbols s and t. An example is the formula $\exists x\bigl[\mathrm{E}(\mathrm{s}, x) \wedge \mathrm{E}(x, \mathrm{t})\bigr]$. A *$\tau$-structure* is a tuple $(V, E, s, t)$ consisting of a graph $(V, E)$ and two vertices $s, t \in V$. A $\tau$-structure is a *model* of a $\tau$-formula if the formula holds when we interpret the relation symbol E as the edge relation $E$ and the constant symbols s and t as the vertices $s$ and $t$. For example, the $\tau$-formula $\exists x\bigl[\mathrm{E}(\mathrm{s}, x) \wedge \mathrm{E}(x, \mathrm{t})\bigr]$ is a model of every $\tau$-structure $(V, E, s, t)$ in which there is a path (more precisely, a walk) from $s$ to $t$ in the graph $(V, E)$ of length exactly 2. The language $A$ is *first-order definable* if there exists a $\tau$-formula $\phi$ such that $\langle V, E, s, t\rangle \in A$ if and only if $(V, E, s, t)$ is a model of $\phi$.

THEOREM 3.1. *For each $k$,* $\text{REACH}_{\alpha\leq k}$ *is first-order definable.*

*Proof.* Let $k \geq 1$ be fixed. We give a stepwise construction of a formula $\phi_{\text{reach},\alpha\leq k}$ such that $(V, E, s, t) \models \phi_{\text{reach},\alpha\leq k}$ if and only if $\langle V, E, s, t\rangle \in \text{REACH}_{\alpha\leq k}$. Roughly, the formula $\phi_{\text{reach},\alpha\leq k}$ will say "$\alpha(G) \leq k$, and it is not the case that condition 3 of Theorem 2.3 holds for $s$ and $t$."

Let $\phi_{\text{distinct}}(v_1, \ldots, v_m) \equiv \bigwedge_{i\neq j}[v_i \neq v_j]$. This formula expresses that vertices are distinct. The property "$\alpha(G) \leq k$" can be expressed as

$$\phi_{\alpha\leq k} \equiv (\forall v_1, \ldots, v_{k+1})\Big[\phi_{\text{distinct}}(v_1, \ldots, v_{k+1}) \to \bigvee_{i\neq j} \mathrm{E}(v_i, v_j)\Big].$$

The next two formulas express that a vertex $v$ or a set $\{v_1, \ldots, v_m\}$ of vertices 2-dominates a vertex $u$:

$$\phi_{\text{2-dom}}(v, u) \equiv v = u \vee \mathrm{E}(v, u) \vee (\exists z)\big[\mathrm{E}(v, z) \wedge \mathrm{E}(z, u)\big],$$
$$\phi_{\text{2-dom}}(v_1, \ldots, v_m, u) \equiv \phi_{\text{2-dom}}(v_1, u) \vee \cdots \vee \phi_{\text{2-dom}}(v_m, u).$$

Since $\beta_2(G) \leq \alpha(G) \leq k$, condition 3 of Theorem 2.3 can be expressed as

$$\begin{aligned}\phi_{\text{condition}} \equiv\ & (\exists v_1, \ldots, v_k)\\ & \Big[\phi_{\text{2-dom}}(v_1, \ldots, v_k, \mathrm{s}) \wedge \neg\phi_{\text{2-dom}}(v_1, \ldots, v_k, \mathrm{t})\\ & \quad \wedge (\forall u, v)\big[\big(\phi_{\text{2-dom}}(v_1, \ldots, v_k, u) \wedge \mathrm{E}(u, v) \to \phi_{\text{2-dom}}(v_1, \ldots, v_k, v)\big)\big]\Big].\end{aligned}$$

The desired formula $\phi_{\text{reach},\alpha\leq k}$ is given by $\phi_{\alpha\leq k} \wedge \neg\phi_{\text{condition}}$. Note that its quantifier complexity (nesting depth) is $k + 3$ and that the number of quantifier alternations is three, beginning with a universal quantifier. ☐

COROLLARY 3.2. *The language* REACH *can be decided in space $O\big(\alpha(G)\log n\big)$.*

*Proof.* On input of a coded tuple $\langle G, s, t\rangle$ with $G = (V, E)$, $s, t \in V$, $n := |V|$, and $\alpha := \alpha(G)$, we can compute $\alpha$ in space $O(\alpha \log n)$ by iteratively testing for increasing values of $\alpha$ whether there exists a subset of $\alpha$ many vertices that is independent. Once we know $\alpha$, we check whether $(V, E, s, t) \models \phi_{\text{reach},\alpha\leq k}$ for $k := \alpha$. This check can also be performed in space $O(\alpha \log n)$, since we need $\log n$ bits to iterate over all possible assignments of a variable bound by a quantifier and since the number of nested quantifiers in $\phi_{\text{reach},\alpha\leq k}$ is $k + 3$. ☐

Theorem 3.1 can easily be extended to the following larger class of graphs: Define the *$q$-independence number* $\alpha_q(G)$ of a graph $G$ as the maximum size of a $q$-independent set in $G$, which is a vertex subset such that there is no path (in all of $G$) of length at most $q$ between any two vertices in this subset. Then reachability in graphs with $\alpha_q(G) \leq k$ is first-order definable for all $k, q \in \mathbb{N}$.

**3.2. Infinite version of the reachability problem.** In this subsection we study the class $\text{REACH}^{\infty}_{\alpha\leq k}$ and show that the first-order definability of $\text{REACH}_{\alpha\leq k}$ does not carry over to $\text{REACH}^{\infty}_{\alpha\leq k}$. This class contains all triples $(G, s, t)$ such that $G$ is a (possibly infinite) graph with $\alpha(G) \leq k$ in which there is a path from $s$ to $t$.

Let us fix the model-theoretic notations and terminology. Once more, we restrict our attention to the signature $\tau = (\mathrm{E}^2, \mathrm{s}, \mathrm{t})$ of graphs with two designated vertices. The notions of $\tau$-structures, first-order $\tau$-formulas, and models are defined as before. A class $K$ of $\tau$-structures, i.e., a class of possibly infinite graphs, each together with two designated vertices, is called *elementary (over finite structures)* if there exists a first-order $\tau$-formula $\phi$ such that for every (finite) $\tau$-structure $(V, E, s, t)$ we have $(V, E, s, t) \models \phi$ if and only if $(V, E, s, t) \in K$. A class $K$ of $\tau$-structures is $\Delta$-*elementary* if there exists a set $\Phi$ of first-order $\tau$-formulas such that for every $\tau$-structure $(V, E, s, t)$ we have $(V, E, s, t) \models \phi$ for all $\phi \in \Phi$ if and only if $(V, E, s, t) \in K$. Some authors use "finitely axiomatizable" instead of "elementary." With these definitions, Theorem 3.1 states that $\textsc{reach}^{\infty}_{\alpha \le k}$ is elementary over finite structures for all $k$. This is no longer true for infinite structures, as the following theorem shows.

Theorem 3.3. *$\textsc{reach}^{\infty}_{\alpha \le k}$ is not $\Delta$-elementary for any $k$.*

*Proof.* The proof follows the standard pattern of proofs applying the compactness theorem. Assume that there exist a set $\Phi$ of first-order $\tau$-formulas and a number $k \ge 1$ such that $(V, E, s, t) \models \Phi$ if and only if $(V, E, s, t) \in \textsc{reach}^{\infty}_{\alpha \le k}$. For each $n \in \mathbb{N}$ let $\psi_n$ be a formula that is satisfied by a graph if there is a path from $s$ to $t$ of length exactly $n$. Consider the set $\Psi := \Phi \cup \{\neg\psi_0, \neg\psi_1, \neg\psi_2, \neg\psi_3, \dots\}$. We claim that every finite $\Psi_0 \subseteq \Psi$ has a model $(V, E, s, t)$. To see this, let $n$ be large enough such that for all $i \ge n$ we have $\neg\psi_i \notin \Psi_0$ and define a tournament $G = (V, E)$ by $V := \{1, \dots, n+1\}$ and $(i, j) \in E$ if and only if $j \le i + 1$. Then $\alpha(G) = 1 \le k$ and the shortest path from $s := 1$ to $t := n + 1$ has length $n$. Thus $(V, E, s, t)$ is a model of $\Psi_0$. By the compactness theorem, $\Psi$ has a model $(V', E', s', t')$ since every finite subset of $\Psi$ has a model. Since this model satisfies $\neg\psi_n$ for all $n$, there cannot be a path of finite length from $s'$ to $t'$ in $G' = (V', E')$. Thus $\Phi \subseteq \Psi$ has a model that is not an element of $\textsc{reach}^{\infty}_{\alpha \le k}$. $\square$

**3.3. Succinct version of the reachability problem.** Up to now, we have not addressed the question of how we encode graphs. Varying the encoding method typically has no effect on the complexity of graph problems, but this is no longer true if we use *succinct graph representations.* In this subsection we show that the complexity of $\textsc{reach}_{\alpha \le k}$ jumps from first-order definable to $\Pi^{\mathrm{P}}_2$-complete if we use succinct representations instead of the usual ones (such as adjacency matrices).

Succinctly represented graphs are given implicitly via a description in some description language. Since succinct representations allow the encoding of large graphs by small codes, numerous graph properties are (provably) harder to check for succinctly represented graphs than for graphs coded in the usual way. Papadimitriou and Yannakakis [23] and Wagner [36] have shown that most interesting graph problems become PSPACE-complete or even NEXP-complete for succinctly represented graphs. For tournaments, Umans [33] has shown that the succinct version of the dominating set problem is $\Sigma^{\mathrm{P}}_2$-complete. There are other problems that become complete for levels of the polynomial hierarchy if the instances are succinctly represented by circuits. Schaefer has shown [26] that the problem of determining the Vapnik–Chervonenkis dimension of a succinctly represented family of subsets of a finite set is $\Sigma^{\mathrm{P}}_3$-complete. For an overview of results of this type, see the survey of Schaefer and Umans [27]. Succinct representations are known to be closely related to the concept of leaf languages; see Borchert and Lozano [3] and Veith [34].

The following formalization of succinct graph representations follows Galperin and Wigderson [10], but others are also possible [11, 36].

Definition 3.4. *A* succinct representation *of a graph $G = \big(\{0, 1\}^n, E\big)$ is a $2n$-input circuit $C$ such that for all $u, v \in \{0, 1\}^n$ we have $(u, v) \in E$ exactly if $C(uv) = 1$.*

The circuit tells us, for any two vertices of the graph, whether there is a directed edge between them or not. Note that $C$ will have size at least $2n$ since it has $2n$ input gates.

Having defined succinct versions of *graphs*, we can next define succinct versions of *languages*. Our definition is not the most general possible; we focus on succinct versions of languages that contain (the standard binary encoding of) graphs together with vertices or numbers (numbers are useful for the distance problem). We represent only the graphs succinctly; the vertex codes and numbers are left unchanged.

DEFINITION 3.5. *Let* $A \subseteq \bigl\{\langle G, b_1, \dots, b_m\rangle \mid G = (V, E)$ *is a finite graph,* $b_i \in V$ *or* $b_i \in \mathbb{N}$ *for all* $i\bigr\}$. *Then* SUCCINCT-$A$ *is the set of all* $\langle C, b_1, \dots, b_m\rangle$ *such that* $C$ *is a succinct representation of a graph* $G$ *with* $\langle G, b_1, \dots, b_m\rangle \in A$.

THEOREM 3.6. *For each* $k$, SUCCINCT-REACH$_{\alpha\le k}$ *is* $\Pi_2^{\mathrm{P}}$*-complete.*

*Proof.* Our first aim is to show SUCCINCT-REACH$_{\alpha\le k} \in \Pi_2^{\mathrm{P}}$. Let $\langle C, s, t\rangle$ be an input and let $C$ represent a graph $G = (V, E)$ with $V = \{0,1\}^n$. Note that $\log_2 |V| = n$. We first check whether $\alpha(G) \le k$ holds, which can easily be done using a coNP-machine. We then check whether there is a path from $s$ to $t$ in $G$. By Theorem 2.3 this is the case if *for all* sets $D_1 \subseteq \{0,1\}^n$ of size at most $\beta_1(G)$ either $s \notin \mathrm{dom}_1(D_1)$ or $t \in \mathrm{dom}_1(D_1)$ or $\mathrm{dom}_1(D_1)$ is not closed under reachability; i.e., *there exist* vertices $u \in \mathrm{dom}_1(D_1)$ and $v \in \{0,1\}^n \setminus \mathrm{dom}_1(D_1)$ such that $C(uv) = 1$. Since $\beta_1(G) \le \lceil \log_c 2^n\rceil = \lceil n \log_c 2\rceil$, the size of each $D_1$ that needs to be checked is linear in $n$, and testing for membership in $\mathrm{dom}_1(D_1)$ can be performed in polynomial time. Putting it all together, we see that the "for all... exists..."-test is a $\Pi_2^{\mathrm{P}}$-algorithm.

Our second aim is to prove that even the reachability problem for tournaments, SUCCINCT-REACH$_{\mathrm{tourn}}$, is $\le_{\mathrm{fo}}$-hard for $\Pi_2^{\mathrm{P}}$. Let $L \in \Pi_2^{\mathrm{P}}$ by any language. By the quantifier characterization of the polynomial hierarchy [37] there exists a polynomial-time-decidable ternary relation $R$ and a constant $d$ such that

$$L = \bigl\{x \mid \forall y \in \{0,1\}^{|x|^d}\ \exists z \in \{0,1\}^{|x|^d}\ R(x, y, z)\bigr\}.$$

To simplify the following presentation, we assume that for every $x$ there exists some $z \in \{0,1\}^{|x|^d}$ such that $R\bigl(x, 1^{|x|^d}, z\bigr)$ holds. In other words, we assume that the "last" $y$ is not important for deciding whether $x \in L$ holds.

The desired $\le_{\mathrm{fo}}$-reduction from $L$ to SUCCINCT-REACH$_{\mathrm{tourn}}$ is obtained in three steps.

1. We describe a mapping of inputs $x$ to exponentially large tournaments $G = (V, E)$ and vertices $s, t \in V$ such that $x \in L$ if and only if $(G, s, t) \in$ REACH$_{\mathrm{tourn}}$.
2. We explain how a succinct representation of the highly structured tournament $G$ can be computed in polynomial time.
3. We use an argument of Veith [34] to show that the polynomial-time computation can be replaced by a first-order query.

For the first step, the construction of the tournament $G$, let $n$ denote the length of $x$ and let $\ell := n^d$. The vertex set of $G$ is $V = \{0,1\}^{2\ell}$. The first $\ell$ bits of a vertex $v \in V$ will be called its *row*, and the last $\ell$ bits its *column*. In other words, the vertices are arranged in a big grid consisting of $2^\ell$ rows and $2^\ell$ columns. All vertices on the same row are connected such that they form a strongly connected subtournament of $G$. Let us say that the lexicographically smallest row is the topmost row and that the lexicographically largest row is the bottommost row.

Edges between different rows generally point "upward," i.e., from rows farther down to rows farther up. The only exception are edges between a vertex $v = yz$ on row $y$ and column $z$ and the vertex on the same column in the row directly below. Such an edge points "downward" if $R(x, y, z)$. The source $s$ is the upper-left corner of the grid, i.e., the vertex $s = 0^\ell 0^\ell$. The target $t$ is the lower-left corner, i.e., the

$$\begin{tikzcd}
s \arrow[r, thick] & y_1z_2 \arrow[r, thick] & y_1z_3 \arrow[r] \arrow[d, thick] & y_1z_4 \arrow[r] & y_1z_5 \arrow[r] & y_1z_6 \arrow[r] & y_1z_7 \arrow[r] & y_1z_8 \arrow[lllllll, bend left] \\
y_2z_1 \arrow[r] & y_2z_2 \arrow[r] \arrow[d] & y_2z_3 \arrow[r] \arrow[d, thick] & y_2z_4 \arrow[r] & y_2z_5 \arrow[r] & y_2z_6 \arrow[r] & y_2z_7 \arrow[r] & y_2z_8 \arrow[lllllll, bend left] \\
y_3z_1 \arrow[r] & y_3z_2 \arrow[r] & y_3z_3 \arrow[r, thick] & y_3z_4 \arrow[r, thick] & y_3z_5 \arrow[r, thick] & y_3z_6 \arrow[r] \arrow[d, thick] & y_3z_7 \arrow[r] \arrow[d] & y_3z_8 \arrow[lllllll, bend left] \arrow[d] \\
y_4z_1 \arrow[r, thick] & y_4z_2 \arrow[r, thick] & y_4z_3 \arrow[r, thick] & y_4z_4 \arrow[r] \arrow[d, thick] & y_4z_5 \arrow[r] & y_4z_6 \arrow[r, thick] & y_4z_7 \arrow[r, thick] & y_4z_8 \arrow[lllllll, bend left, thick] \\
y_5z_1 \arrow[r] \arrow[d] & y_5z_2 \arrow[r] & y_5z_3 \arrow[r] & y_5z_4 \arrow[r] \arrow[d, thick] & y_5z_5 \arrow[r] & y_5z_6 \arrow[r] & y_5z_7 \arrow[r] & y_5z_8 \arrow[lllllll, bend left] \arrow[d] \\
y_6z_1 \arrow[r] \arrow[d] & y_6z_2 \arrow[r] \arrow[d] & y_6z_3 \arrow[r] \arrow[d] & y_6z_4 \arrow[r] \arrow[d, thick] & y_6z_5 \arrow[r] \arrow[d] & y_6z_6 \arrow[r] & y_6z_7 \arrow[r] & y_6z_8 \arrow[lllllll, bend left] \\
y_7z_1 \arrow[r] & y_7z_2 \arrow[r] & y_7z_3 \arrow[r] & y_7z_4 \arrow[r, thick] & y_7z_5 \arrow[r, thick] & y_7z_6 \arrow[r] \arrow[d, thick] & y_7z_7 \arrow[r] & y_7z_8 \arrow[lllllll, bend left] \\
t \arrow[r] & y_8z_2 \arrow[r] & y_8z_3 \arrow[r] & y_8z_4 \arrow[r] & y_8z_5 \arrow[r] & y_8z_6 \arrow[r, thick] & y_8z_7 \arrow[r, thick] & y_8z_8 \arrow[lllllll, bend left, thick]
\end{tikzcd}$$

FIG. 3.1. *Example of a tournament $G$ constructed in the proof of Theorem* 3.6 *for $\ell = 3$. There are $2^\ell$ rows $y_1 = 000$, $y_2 = 001$, ..., $y_8 = 111$ and $2^\ell$ columns $z_1 = 000$, $z_2 = 001$, ..., $z_8 = 111$. Most arrows have been omitted for clarity; the missing arrows point upward for vertices on different rows and right for vertices on the same row. Downward arrows correspond to positions for which the predicate $R$ is true. For example, the downward arrow between the first two rows indicates that $R(x, y_1, z_3)$ holds. A path from $s$ to $t$, which is indicated in bold, exists since for each $y_i$ there exists some $z_j$ such that $R(x, y_i, z_j)$ holds.*

vertex $1^\ell 0^\ell$. An example of a tournament $G$ constructed in this way is depicted in Figure 3.1.

We claim that $(G, s, t) \in \text{REACH}_{\text{tourn}}$ if and only if $x \in L$. First, observe that $G$ is a tournament. Next, note that from each row $y$ one can go directly (at best) only one row down since all edges between nonneighboring rows point upward. Since all vertices on the same row are connected, if we can reach a vertex $v$ on row $y$, we can reach any vertex on the row directly below if and only if $R(x, y, z)$ holds for some $z \in \{0,1\}^\ell$. So we can "go all the way from the source down to the target" if for all $y \in \{0,1\}^\ell$ there exists a $z \in \{0,1\}^\ell$ such that $R(x, y, z)$. Recall that we assumed that for $y = 1^\ell$ there always exists some $z \in \{0,1\}^\ell$ with $R(x, y, z)$.

For the second step we argue that a succinct representation $C$ of $G$ can be computed in time polynomial in $n$. The circuit $C$ has $4\ell$ input gates. Through these gates two vertices $v, v' \in \{0,1\}^{2\ell}$ are fed into the circuit. In order to decide whether there is an edge pointing from $v$ to $v'$ in $G$, the circuit disassembles $v$ and $v'$ into their row parts $y$ and $y'$ and their column parts $z$ and $z'$. If the vertices lie on the same row, i.e., if $y = y'$, the circuit normally outputs 1 if $z$ is lexicographically smaller than $z'$ *except* for the edge from column $0^\ell$ to column $1^\ell$, which is turned around. This ensures that the vertices on each row form a strongly connected subtournament. If $y \neq y'$, the circuit normally outputs 1 if $y'$ is lexicographically smaller than $y$, which ensures that edges between different rows point "upward." The exception occurs when $z = z'$ and $y$ and $y'$ are only one row apart. In this case, the circuit outputs 1 if $R(x, y, z)$ and $y'$ is the lexicographic successor of $y$. For the check whether $R(x, y, z)$ holds, $C$ contains a subcircuit that evaluates $R$. Since $R$ is polynomial-time decidable, this subcircuit can be obtained in polynomial time. Hence, it takes only polynomial time to write down the code of a circuit $C$ that performs all of the above computations.

For the final step we invoke an observation of Veith [34], who showed that every polynomial-time many-one reduction to a succinctly represented problem can be turned into a $\leq_{\text{fo}}$-reduction to the same problem by "shifting" most of the computation done by the polynomial-time reduction machine into the circuit that succinctly represents a problem instance. However, Veith points out that this result is somewhat sensitive to the exact definition of succinctness and applies only if the polynomial-time reduction's output does not contain any parts that are not succinctly represented—such as the source and target in our definition or the length of the encoded bitstring in Borchert and Lozano's definition [3].

Although we cannot apply Veith's result directly, we can use the following key observation, which is implicit in Veith's paper: *Let $f$ be a polynomial-time computable function that outputs codes of circuits. Then there exists a first-order query $f'$ that also outputs codes of circuits such that for all $x$ the circuits encoded by $f(x)$ and $f'(x)$ compute the same function.* We can now assemble the desired $\leq_{\text{fo}}$-reduction from $L$ to $\textsc{succinct-reach}_{\text{tourn}}$ as follows: By the observation, there exists a first-order query that maps the input $x$ to a circuit $C'$ that, just like $C$, is a succinct representation of $G$. The source $s = 0^\ell 0^\ell$ and the target $t = 1^\ell 0^\ell$ can clearly be computed by first-order queries. Putting it all together, we get the desired reduction. $\square$

The tournament constructed in the above proof is a strong tournament (a strongly connected tournament) if and only if there is a path from $s$ to $t$. This proves the following corollary.

Corollary 3.7. $\textsc{succinct-strong-tournament}$ *is* $\Pi_2^{\text{P}}$*-complete.*

As mentioned earlier, succinct representations are closely related to leaf classes. Borchert and Lozano [3] and Veith [34] have shown that $\textsc{succinct-}A$ is complete for the balanced leaf class $\text{BLEAF}^{\text{P}}(A)$. The above completeness results can thus be stated equivalently as $\text{BLEAF}^{\text{P}}(\textsc{reach}_{\alpha \leq k}) = \Pi_2^{\text{P}}$ and $\text{BLEAF}^{\text{P}}(\textsc{strong-tournament}) = \Pi_2^{\text{P}}$.

## 4. Complexity of the construction problem.

In this section we show that for graphs with bounded independence number we not only can tell in logarithmic space whether a path exists between two vertices, but also can construct such a path. While it seems difficult to construct the *shortest* path in logarithmic space (by the results of the next section this is impossible unless $\text{L} = \text{NL}$), it is possible to find a path that is *approximately* as long as the shortest path. Even better, there exists a *logspace approximation scheme* for constructing paths whose lengths are as close to the length of the shortest path as we would like. This logspace approximation scheme can be obtained from the following theorem, which relates the space complexity of the reachability problem to the independence number $\alpha$ and to the desired approximation ratio $r = 1 + 1/m$, by fixing the maximum independence number and the ratio.

Theorem 4.1. *There exists a deterministic Turing machine $M$ with read-only access to the input tape and write-only access to the output tape with the following properties: On input $\langle G, s, t, m\rangle$, where $G = (V, E)$ is a graph, $s, t \in V$, $m \geq 1$, $n := |V|$, and $\alpha := \alpha(G)$,*

1. *if $\langle G, s, t\rangle \in \textsc{reach}$, then $M$ outputs a path from $s$ to $t$ of length at most*

$(1+1/m)\,\mathrm{d}(s,t)$ *and uses* $O\big((\alpha+\log m)\log n\big)$ *space on the work tape; and*

2. *if* $\langle G,s,t\rangle \notin$ REACH, *it outputs "no path exists" and uses* $O(\alpha\log n)$ *space on the work tape.*

The theorem's proof uses three lemmas. The first of these lemmas is a "constructive version" of Savitch's theorem [25]. Similarly to Theorem 4.1, the lemma relates the space complexity of the reachability problem to a graph parameter, namely, to the length of the shortest path.

LEMMA 4.2. *There exists a deterministic Turing machine* $M_{\text{Savitch}}$ *with read-only access to the input tape and write-only access to the output tape with the following properties: On input* $\langle G,s,t\rangle$, *where* $G=(V,E)$ *is a graph,* $s,t\in V$, *and* $n:=|V|$,

1. *if* $\langle G,s,t\rangle \in$ REACH, *then* $M_{\text{Savitch}}$ *outputs a shortest path from* $s$ *to* $t$ *and uses* $O\big(\log \mathrm{d}(s,t)\log n\big)$ *space on the work tape; and*

2. *if* $\langle G,s,t\rangle \notin$ REACH, *it outputs "no path exists" and uses* $O(\log^2 n)$ *space on the work tape.*

*Proof.* We augment Savitch's algorithm [25] by a construction procedure that outputs paths. The main difficulty is that if there are several shortest paths, then the procedure must "decide on one of them" and must do so "within the recursion." Since it will be useful later, the construction procedure outputs the list of edges on the shortest path, not only the vertices themselves.

Let *reachable*$(u,v,\ell)$ be Savitch's procedure for testing whether there is a path from $u$ to $v$ of length at most $\ell$: For $\ell=1$, it checks whether $(u,v)\in E$. For larger $\ell$, it checks whether for some vertex $z$ both the calls *reachable*$(u,z,\lfloor\ell/2\rfloor)$ and *reachable*$(z,v,\ell-\lfloor\ell/2\rfloor)$ succeed; see Figure 4.1 for pseudocode. As noted by Savitch, since we can reuse space, we can compute *reachable*$(u,v,\ell)$ in space $O(\log\ell\log n)$.

We next define a procedure *construct-path*$(u,v,\ell)$ that writes the edges of a path of length $\ell$ from $u$ to $v$ onto an output tape, provided *reachable*$(u,v,\ell)$ holds. For $\ell=1$, *construct-path* outputs $(u,v)$. For larger $\ell$, it finds the first vertex $z$ for which both the calls *reachable*$(u,z,\lfloor\ell/2\rfloor)$ and *reachable*$(z,v,\ell-\lfloor\ell/2\rfloor)$ succeed. For this vertex $z$ it first calls *construct-path*$(u,z,\lfloor\ell/2\rfloor)$ and then *construct-path*$(z,v,\ell-\lfloor\ell/2\rfloor)$; see Figure 4.1 once more.

The machine $M_{\text{Savitch}}$ iteratively calls *reachable*$(s,t,\ell)$ for increasing values of $\ell$. For the first value $\ell$ for which this test succeeds, it calls *construct-path*$(s,t,\ell)$ and quits. If the tests do not succeed for any $\ell\le n$, it outputs "no path exists." ∎

LEMMA 4.3. *Let* $G=(V,E)$ *be a graph and let* $u,v,v'\in V$ *be vertices with* $\mathrm{d}(u,v)=\mathrm{d}(u,v')$. *Let* $T$ *be the set of edges on the paths* $p$ *and* $p'$ *output by* $M_{\text{Savitch}}$ *on the inputs* $\langle G,u,v\rangle$ *and* $\langle G,u,v'\rangle$, *respectively. Then* $(V,T)$ *is a tree, i.e., the paths do not "split and join again."*

*Proof.* We prove the claim by induction on the distance $d=\mathrm{d}(u,v)=\mathrm{d}(u,v')$. The claim is true for distance 1. For the inductive step, consider the two vertices $z$ and $z'$ on the paths $p$ and $p'$ at distance $\lfloor d/2\rfloor$ from $u$. If $z=z'$, then the paths $p$ and $p'$ are identical up to $z$, and they form a tree after the vertex $z$ by the induction hypothesis applied to the three vertices $z$, $v$, and $v'$ for the distance $d-\lfloor d/2\rfloor<d$. If $z\ne z'$, then the parts of $p$ and $p'$ leading to $z$ and $z'$ form a tree by the induction hypothesis applied to $u$, $z$, and $z'$ for the distance $\lfloor d/2\rfloor<d$. Furthermore, $z\ne z'$ implies that the rest of the paths are completely disjoint since $v$ is not reachable from $z'$ and $v'$ is not reachable from $z$ (otherwise $M_{\text{Savitch}}$ would not have chosen two different midpoints $z$ and $z'$). ∎

LEMMA 4.4. *There exists a logspace-computable function that maps every input* $\langle G,s,t\rangle\in$ REACH$_{\text{forest}}$ *to the shortest path from* $s$ *to* $t$ *in* $G$ *and all other inputs to "no path exists."*

```
PROCEDURE reachable($u, v, \ell$)
INPUT: Vertices $u, v \in V$, a distance $\ell$
RETURN VALUE: Does a path exist?
1  if $\ell = 0$ then return yes if $u = v$ and no if $u \neq v$
2  else if $\ell = 1$ then return yes if $(u, v) \in E$ and no if $(u, v) \notin E$
3  else
4      for $i \leftarrow 1$ to $n$ do
5          if reachable($u, v_i, \lfloor \ell/2 \rfloor$) and reachable($v_i, v, \ell - \lfloor \ell/2 \rfloor$) then
6              return yes
7  return no
```

```
PROCEDURE construct-path($u, v, \ell$)
INPUT: Vertices $u, v \in V$, a distance $\ell$
OUTPUT: Edges on a path from $u$ to $v$
PRECONDITION: $1 \leq \mathrm{d}(u, v) \leq \ell$
1  if $\ell = 1$ then output "($u$,$v$)"; exit
2  else
3      for $i \leftarrow 1$ to $n$ do
4          if reachable($u, v_i, \lfloor \ell/2 \rfloor$) and reachable($v_i, v, \ell - \lfloor \ell/2 \rfloor$) then
5              call construct-path($u, v_i, \lfloor \ell/2 \rfloor$)
6              call construct-path($v_i, v, \ell - \lfloor \ell/2 \rfloor$)
7              exit
```

FIG. 4.1. *Savitch's algorithm reachable*$(u, v, \ell)$ *decides whether* $\mathrm{d}(u, v) \leq \ell$ *holds in some graph* $G = (V, E)$ *with* $V = \{v_1, \dots, v_n\}$. *Its "constructive" version construct-path*$(u, v, \ell)$ *outputs such a path, more precisely, the list of edges on this path. Here and in the following we do not explicitly list the immutable* $G$ *as an input parameter. Rather,* $G$ *is treated as a "global constant."*

*Proof.* The problem $\text{REACH}_{\text{forest}}$ is L-complete as shown in [4]. In order to compute the shortest path from $s$ to $t$ we iterate the following instructions, starting at $v = s$: For each successor $v'$ of the current vertex $v$, we check whether $t$ is reachable from $v'$. There is exactly one vertex $v'$ for which this test succeeds. We output this $v'$ or, if preferred, the edge $(v, v')$, make $v'$ the new current vertex, and repeat the procedure until we reach $t$. □

*Proof of Theorem* 4.1. Let an input $\langle G, s, t, m \rangle$ be given. Let $G = (V, E)$ and $n := |V|$. By Corollary 3.2 we can check in space $O(\alpha \log n)$ whether $\langle G, s, t \rangle \in \text{REACH}$ holds and output "no path exists" if this is not the case. Otherwise we proceed as follows.

Our algorithm is the composition of three functions. Each function takes the input of the previous function and maps it to an output, using space $O\big((\alpha + \log m) \log n\big)$ on the work tape. More precisely, the first two functions need space $O\big((\alpha + \log m) \log n\big)$, the last one only $O(\log n)$. Although the intermediate outputs are too large "to be written down" on the work tape of the composed machine, we can use the standard trick of composing logspace-computable functions: Whenever one of the functions needs a bit of the output of the previous function, we recalculate this bit "on the fly." This allows us to compute the composed function in space $O\big(2(\alpha + \log m) \log n + \log n\big) = O\big((\alpha + \log m) \log n\big)$.

*The first function: Construction of the* $U_i$. The first function takes the original input $\langle G, s, t, m \rangle$ and maps it to a sequence $U_1, U_2, \dots, U_\ell \subseteq V$ of vertex sets with $U_1 = \{s\}$ and $U_\ell = \{t\}$. For the construction of $U_i$ we access only $U_{i-1}$ and use space $O\big((\alpha + \log m) \log n\big)$. Once we have constructed $U_i$, we erase $U_{i-1}$ from the work tape and reuse the space it had occupied.

```
PROCEDURE compute-next-dominating-set($U_{i-1}$)
INPUT: Previous set $U_{i-1}$
RETURN VALUE: Next set $U_i$
1  foreach $U \subseteq V$ with $|U| \leq \alpha$ do
2      foreach $u \in U$ do
3          if not $\mathrm{d}(U_{i-1}, u) = 2m + 2$ then
4              continue with next $U$ on line 1
5      foreach $v \in V$ do
6          if $\mathrm{d}(U_{i-1}, v) = 2m + 2$ and not $\mathrm{d}(U, v) \leq 2$ then
7              continue with next $U$ on line 1
8      return $U$
```

```
PROCEDURE output-all-U
OUTPUT: The list $(U_1, \dots, U_\ell)$
PRECONDITION: $t$ is reachable from $s$
1  $i \leftarrow 1$
2  $U \leftarrow \{s\}$
3  output "$U_1 = \{s\}$"
4  while not $\mathrm{d}(U, t) \leq 2m + 2$ do
5      $U \leftarrow$ compute-next-dominating-set($U$)
6      $i \leftarrow i + 1$
7      output "$U_i = U$"
8  $\ell \leftarrow i + 1$
9  output "$U_\ell = \{t\}$"
```

FIG. 4.2. *Pseudocode for the computation of the sets $U_i$. To store $U$ and $U_{i-1}$, we need space $O(\alpha \log n)$. To perform the checks $\mathrm{d}(U_{i-1}, u) = 2m + 2$ and $\mathrm{d}(U, t) \leq 2m + 2$, we need extra space $O(\log m \log n)$. In total, the procedure outputs all $U_i$ using space $O\big((\alpha + \log m) \log n\big)$.*

The set $U_i$ is obtained from $U_{i-1}$ as follows: If $\mathrm{d}(U_{i-1}, t) \leq 2m+2$, let $U_i := \{t\}$. Otherwise let $S_i := \{v \in V \mid \mathrm{d}(U_{i-1}, v) = 2m + 2\}$ be the set of all vertices that have distance exactly $2m + 2$ from $U_{i-1}$. Choose $U_i \subseteq S_i$ as a 2-dominating vertex subset of size at most $\alpha$ of the graph $G' := \big(S_i, E \cap (S_i \times S_i)\big)$ induced on the vertices in $S_i$. Since $\alpha(G') \leq \alpha$, such a 2-dominating set $U_i$ exists by Theorem 2.2. We can obtain it in space $O\big((\alpha + \log m) \log n\big)$ as follows: For each subset $U \subseteq V$ of size at most $\alpha$ we check whether all $u \in U$ are in $S_i$—which can be checked in space $O(\log m \log n)$ for each $u$ using the procedure *reachable* from Lemma 4.2—and then check whether all elements of $S_i$ are 2-dominated by $U$. A set $U$ passing these checks can be used as $U_i$. Pseudocode for the computation of the $U_i$ is given in Figure 4.2.

The sets $U_i$ have the following properties for $i \in \{2, \dots, \ell - 1\}$:

1. All elements of $U_i$ are reachable from $s$.
2. $|U_i| \leq \alpha$.
3. $\mathrm{d}(U_{i-1}, u) = 2m + 2$ for all $u \in U_i$.
4. $\mathrm{d}(U_i, t) \leq \mathrm{d}(U_{i-1}, t) - 2m$ and hence $\mathrm{d}(U_i, t) \leq \mathrm{d}(s, t) - 2m(i - 1)$.

To see that the last property holds, note that $\mathrm{d}(U_i, t) \leq \mathrm{d}(S_i, t) + 2$ and that $\mathrm{d}(S_i, t) = \mathrm{d}(U_{i-1}, t) - 2m - 2$. For $i = \ell$, the first two properties are also true, and the third becomes $\mathrm{d}(U_{i-1}, t) \leq 2m + 2$.

Intuitively, in each iteration we reduce the distance between $U_i$ and $t$ by at least $2m$, and each $U_{i-1}$ can be connected to the next $U_i$ by a path of length $2m + 2$; see also Figure 4.3. It remains to explain how to connect the $U_i$'s correctly.

*The second function: Construction of the tree.* In order to output the desired path from $s$ to $t$ of length at most $(1 + 1/m)\,\mathrm{d}(s, t)$, we first construct a tree $T$ that

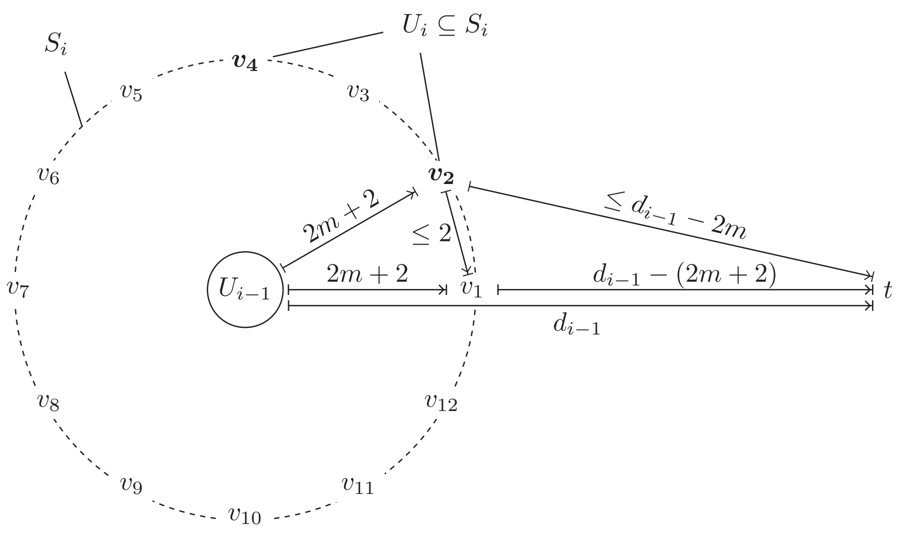


FIG. 4.3. *Visualization of the idea behind the construction of the set $U_i$, whose elements are shown in bold, based on the set $U_{i-1}$. The set $S_i = \{v_1, \dots, v_{12}\}$ contains all vertices at distance exactly $2m+2$ from $U_{i-1}$. The 2-dominating set $U_i$ for $S_i$ typically does not contain the vertex $v_1$ that is on the shortest path from $U_{i-1}$ to $t$. However, it will contain a vertex $v_2$ that is at most two steps removed from $v_1$. If the distance from $U_{i-1}$ to $t$ is $d_{i-1}$, then the distance from $v_2$ to $t$ is at most $d_{i-1} - 2m$. Thus, investing $2m+2$ steps to get to $v_2$ gets us $2m$ steps nearer to the target.*

contains this path. The shortest path in the tree will be the desired path. The second function takes the list of $U_i$'s computed by the first function as input and outputs the tree $T$.

The tree is the union of forests $F_i$ for $i \in \{2, \dots, \ell\}$. Each $F_i$ contains for each $u \in U_i$ the vertices and edges of a specific shortest path from $U_{i-1}$ to $u$. This path is constructed by calling the machine $M_{\text{Savitch}}$ from Lemma 4.2 on input $\langle G, u', u\rangle$ for the first vertex $u' \in U_{i-1}$ for which $\mathrm{d}(u', u) = \mathrm{d}(U_{i-1}, u)$ holds, i.e., for which $\mathrm{d}(u', u)$ is minimal. Since $\mathrm{d}(U_{i-1}, u) \leq 2m+2$, this call needs space $O(\log m \log n)$. Pseudocode for the construction of the $F_i$ is given in Figure 4.4. An example of a tree constructed in this way is depicted in Figure 4.5.

We claim that each graph $F_i$ is a forest. First, if the paths to two vertices $u_1, u_2 \in U_i$ start at two distinct vertices $u_1', u_2' \in U_{i-1}$, then these paths must be completely disjoint (otherwise we would have chosen the same starting vertex). Thus, $F_i$ can be partitioned into independent subgraphs, each containing exactly the paths originating at one vertex in $U_{i-1}$, and all these paths have the same length $\mathrm{d}(U_{i-1}, U_i)$. By Lemma 4.3 each of the subgraphs is a tree and thus the whole graph $F_i$ is a forest.

Let $T$ be the union of all the forests $F_i$ constructed during the run of the algorithm. This union is a tree since

1. every vertex in the union is reachable from the source by construction and
2. there cannot be two different paths to the same vertex $v$ in $T$ since they would induce, inside one of the forests $F_i$, two different paths to $v$ from different roots of $F_i$.

*The third function: Constructing the shortest path in the tree.* The final function outputs the shortest path from $s$ to $t$ in $T$, using the algorithm from Lemma 4.4. This path passes through all $U_i$. For $i \in \{1, \dots, \ell\}$ let $u_i \in U_i$ be the last vertex of $U_i$ on this path. The total length of the path is given by $\sum_{i=1}^{\ell-1} \mathrm{d}(u_i, u_{i+1})$. We have

```
PROCEDURE output-forest($U_{i-1}, U_i$)
INPUT: Pair $(U_{i-1}, U_i)$
OUTPUT: List of edges in the forest $F_i$
1    $d \leftarrow \mathrm{d}(U_{i-1}, U_i)$
2    foreach $u \in U_i$ do
3        foreach $u' \in U_{i-1}$ do
4            if $\mathrm{d}(u', u) = d$ then
5                call construct-path($u', u, d$)
6                continue with next $u$ on line 2
```

```
PROCEDURE output-tree($U_1, \dots, U_\ell$)
INPUT: The list $(U_1, \dots, U_\ell)$
OUTPUT: List of edges in the tree $T$
1    for $i \leftarrow 2$ to $\ell$ do
2        call output-forest($U_{i-1}, U_i$)
```

FIG. 4.4. *Procedures for computing the $F_i$ and the complete tree $T$. In the inner loop of output-forest, the vertices of $U_{i-1}$ are processed in the same order each time. Each call of the procedure construct-path will output edges of the forest $F_i$. Some edges might be output repeatedly, but this is not a problem for the subsequent computation.*

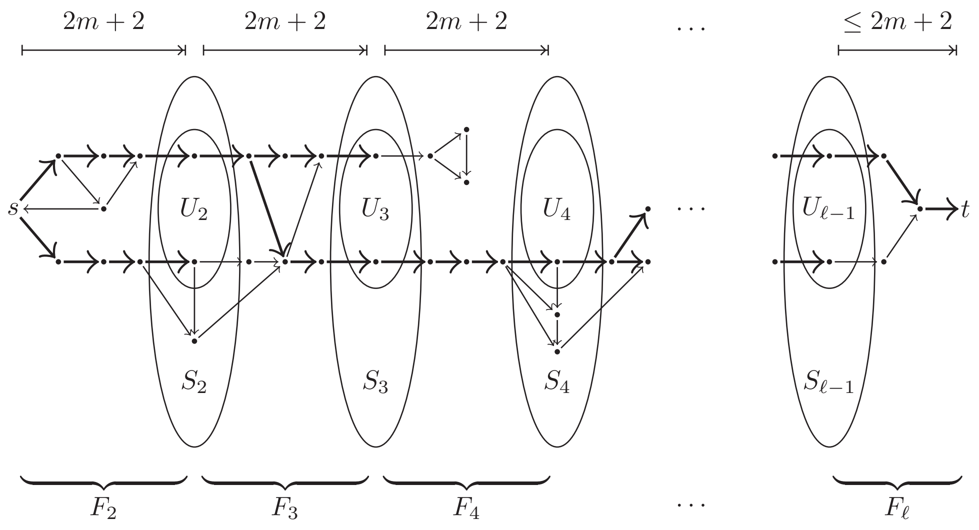


FIG. 4.5. *Example for the construction of the tree $T$, shown in bold, for $m = 1$. It is the union of the forests $F_i$, which connect the vertices from $U_i$ to $U_{i-1}$. For example, the forest $F_2$ consists of the two paths leading from $U_1 = \{s\}$ to the two vertices of $U_2$. The forest $F_3$ consists of the two paths from the top vertex of $U_2$ to the vertices in $U_3$. The approximation algorithm outputs the path from $s$ to $t$ inside the tree $T$. This path is not the shortest path, which follows the bottommost edges, but it is not much longer.*

$\mathrm{d}(u_i, u_{i+1}) = 2m + 2$ for $i \in \{1, \dots, \ell - 2\}$. Thus the total length is

$$\begin{aligned}(2m+2)(\ell-2) + \mathrm{d}(u_{\ell-1}, t) &= (2m+2)(\ell-2) + \mathrm{d}(U_{\ell-1}, t)\\ &\leq (2m+2)(\ell-2) + \mathrm{d}(s,t) - 2m(\ell-2)\\ &= \mathrm{d}(s,t) + 2(\ell-2)\\ &\leq \mathrm{d}(s,t) + \mathrm{d}(s,t)/m.\end{aligned}$$

For the two inequalities, both times we used the last property of $U_{\ell-1}$, by which $\mathrm{d}(U_{\ell-1},t) \leq \mathrm{d}(s,t) - 2m(\ell-2)$ and hence also $2(\ell-2) \leq \mathrm{d}(s,t)/m$. ∎

The space bound from Theorem 4.1 is optimal in the following sense: Suppose we could construct a machine $M'$ that uses space $O(\log^{1-\epsilon} m \log n)$ for some $\alpha$ and achieves the same as $M$ for that particular $\alpha$. This would imply $\text{DISTANCE}_{\text{tourn}} \in \text{DSPACE}[\log^{2-\epsilon} n]$, because $M'$ outputs the *shortest* path for $m = n+1$, and for tournaments we have $\alpha = 1$. The results of the next section show that this would imply $\text{NL} \subseteq \text{DSPACE}[\log^{2-\epsilon} n]$.

**5. Complexity of the distance problem.** In this section we study the complexity of the distance problem for graphs with bounded independence number. This problem asks us to decide whether the distance of two vertices in a graph is smaller than a given input number. We show that this problem is NL-complete even for tournaments and that the succinct version is PSPACE-complete.

The distance problem is closely linked to the problem of constructing a shortest path in a graph: As argued in the introduction, we can *construct* a shortest path in a graph if we have oracle access to the distance problem for this graph. The other way round, we can easily solve the distance problem if we have oracle access to an algorithm that constructs shortest paths. Because of this close relationship, the completeness result dashes any hope of finding a logspace algorithm for constructing shortest paths in tournaments unless L = NL.

THEOREM 5.1. *The problem* $\text{DISTANCE}_{\text{tourn}}$ *is* NL*-complete.*

*Proof.* We show $\text{REACH} \leq_{\text{fo}} \text{DISTANCE}_{\text{tourn}}$. Let an input $\langle G, s, t\rangle$ be given. Our first-order reduction maps this to an instance $\langle G', s', t', d\rangle$ for $\text{DISTANCE}_{\text{tourn}}$. Let $G = (V,E)$ and $n := |V|$. The tournament $G' = (V', E')$ is constructed as follows: The vertex set $V'$ is $\{1,\dots,n\} \times V$. We can think of this vertex set as a grid consisting of $n$ rows and $n$ columns. There is an edge in $G'$ from a vertex $(r_1, v_1)$ to a vertex $(r_2, v_2)$ if and only if one of the following conditions holds (see also Figure 5.1):

1. $r_2 = r_1 + 1$ and $(v_1, v_2) \in E$ or $v_1 = v_2$; i.e., if $v_1$ and $v_2$ are connected in $G$ or if they are the same, then there is an edge leading "downward" between them on adjacent rows.
2. $r_2 = r_1$ and $v_1 < v_2$; i.e., the vertices on the same row are ordered by the linear ordering to which the first-order reduction has access.
3. $r_2 = r_1 - 1$, $(v_1, v_2) \notin E$, and $v_1 \neq v_2$; i.e., if $v_1$ and $v_2$ are not connected in $G$ and if they are different, then there is an edge leading "upward" between them on adjacent rows.
4. $r_2 \leq r_1 - 2$, i.e., all edges spanning at least two rows point "upward."

Clearly, the above conditions for the edge relation $E'$ can be expressed by a first-order formula $\phi_{E'}(r_1, v_1, r_2, v_2)$. The new source $s' := (1, s)$, the new target $t' := (n, t)$, and the distance $d := n - 1$ are also first-order definable.

To see that this reduction works, first assume that there exists a path from $s$ to $t$ in $G$ of length $m \leq n-1$. Let $(s, v_2, \dots, v_m, t)$ be this path. Then $\bigl((1,s), (2,v_2), \dots, (m, v_m), (m+1, t), \dots, (n,t)\bigr)$ is a path in $G'$ of length $n-1$. Second, assume that there exists a path from $s'$ to $t'$ in $G'$ of length $m \leq n-1$. Then $m = n-1$ since any path from the first row to the last row must "brave all rows"—there are no edges that allow us to skip a row. Let $(v'_1, \dots, v'_n)$ be this path. Then $v'_i = (i, v_i)$ for some vertices $v_i \in V$. The sequence $(v_1, \dots, v_n)$ is "almost" a path from $s$ to $t$ in $G$: For each $i \in \{1, \dots, n-1\}$ we have either $v_i = v_{i+1}$ or $(v_i, v_{i+1}) \in E$. Thus, by removing consecutive duplicates and loops, we obtain a path from $s$ to $t$ in $G$. ∎

By the above theorem, DISTANCE and $\text{DISTANCE}_{\text{tourn}}$ are $\leq_{\text{fo}}$-equivalent, while

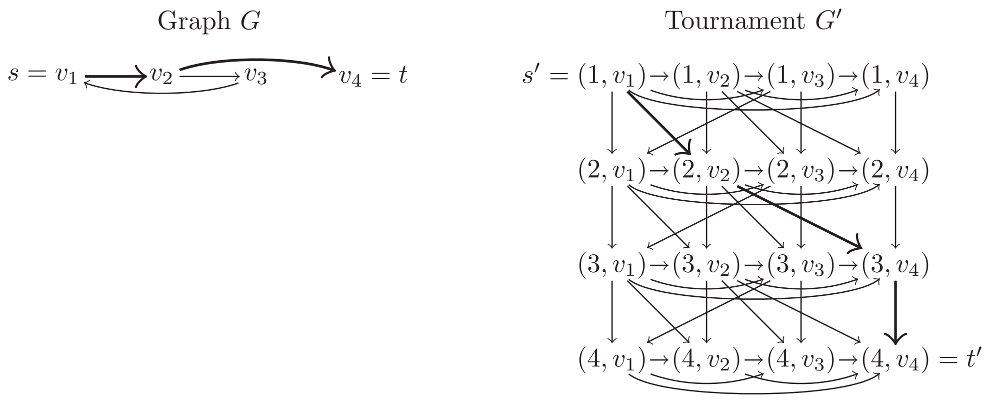


FIG. 5.1. *Example of the construction from Theorem* 5.1. *Arrows have been omitted on the right-hand side for clarity; all missing arrows point upward. The first-order reduction maps an input* $(G, s, t)$*, shown on the left-hand side, to the tournament on the right-hand side. There is a path from* $s$ *to* $t$ *in* $G$ *if and only if there is a path from* $s'$ *to* $t'$ *in* $G'$ *of length* $n - 1 = 3$*. Examples of such paths are shown in bold.*

REACH and REACH$_{\text{tourn}}$ are not. The "complexity jump" that occurs from REACH$_{\text{tourn}}$ to DISTANCE$_{\text{tourn}}$ is reflected by a similar jump for the succinct versions.

THEOREM 5.2. SUCCINCT-DISTANCE$_{\text{tourn}}$ *is* PSPACE-*complete.*

*Proof.* Since DISTANCE$_{\text{tourn}} \in$ NL, we have

$$\text{SUCCINCT-DISTANCE}_{\text{tourn}} \in \text{NPSPACE} = \text{PSPACE}.$$

For the hardness, let $A \in$ PSPACE be an arbitrary language and let $M$ be a polynomial-space machine that accepts $A$. We show $A \leq_{\text{fo}}$ SUCCINCT-DISTANCE$_{\text{tourn}}$. For an input $x$, let $G$ denote the configuration graph of $M$ on input $x$, let $s$ be the initial configuration, let $t$ be the (unique) accepting configuration, and let $n$ be the (exponential) size of $G$'s vertex set. Let $(G', s', t', n - 1)$ be the instance constructed in Theorem 5.1. Then $x \in A$ if and only if $\langle G', s', t', n - 1\rangle \in$ DISTANCE$_{\text{tourn}}$.

To finish the proof, we argue similarly as in the proof of Theorem 3.6: A succinct representation $C$ of $G'$ can be constructed in polynomial time. Invoking Veith's observation yields that a circuit $C'$ computing the same function as $C$ can even be obtained by a first-order query. Next, the source $s'$, the target $t'$, and the distance $d = n - 1$ can also be obtained from $x$ via first-order queries. Putting them together yields the desired first-order reduction from $A$ to SUCCINCT-DISTANCE$_{\text{tourn}}$. □

## 6. Conclusion.

How difficult is it to find paths in graphs with bounded independence number? Our results answer the question in three different ways, depending on exactly what is meant by this question. Checking only whether a path *exists* in a given graph can be done using AC$^0$-circuits. *Constructing* a path between two vertices can be done in logarithmic space. Constructing the *shortest* path in logarithmic space was shown to be impossible, unless L = NL.

These results settle the approximability of the (logspace) optimization problem "shortest paths in graphs with bounded independence number." We have shown that this minimization problem cannot be solved exactly in logarithmic space (unless L = NL), but it can be approximated well: There exists a logspace approximation scheme for it. We pointed out that the space dependency $O\big((\alpha + \log m) \log n\big)$ of our algorithm on the desired approximation ratio $1 + 1/m$ is essentially optimal—any approximation scheme that does substantially better could be used to show the unlikely inclusion

NL $\subseteq$ DSPACE[$\log^{2-\epsilon} n$]. It seems appropriate to call our scheme a "*fully* logspace approximation scheme" in analogy to "fully polynomial-time approximation schemes."

The shortest path problem for tournaments is not the only logspace optimization problem with surprising properties: The distance problem for *undirected* graphs is also NL-complete, while the reachability problem is SL-complete. The proof of the NL-completeness follows the same pattern as our completeness proof for tournaments, except that "upward" edges are omitted and "downward" edges are turned into undirected edges; detailed proofs are given in the technical reports of Toda [30] and Tantau [28]. On the other hand, the distance problem for directed graphs is just as hard as the reachability problem for directed graphs. This shows that, just as in the polynomial-time setting, logspace optimization problems can have different approximation properties, although their underlying decision problems have the same complexity.

The complexity of problems where some parameter—like the graph parameter "independence number"—is fixed is studied extensively in the theory of fixed parameter tractability. Our results can be interpreted as fixed parameter results, although they obviously do not fit directly into the classical framework of polynomial-time fixed parameter tractability.

We would like to recommend the following problems for further research:

1. We do not know whether the quantifier complexity $k + 3$ in the first-order formula for $\text{REACH}_{\alpha \leq k}$ is necessary but conjecture that this is the case. We were able to prove this conjecture for tournaments using an Ehrenfeucht–Fraïssé game played on two appropriate tournaments with 64 vertices. Ehrenfeucht–Fraïssé games [5, 8] seem particularly well-suited for proving matching lower bounds here, since we do not refer to an ordering relation in our first-order formula.

2. In the succinct setting we proved that the problem $\text{SUCCINCT-REACH}_{\alpha \leq k}$ is $\Pi_2^{\text{P}}$-complete for all $k$. In contrast to this, for $q > 1$ our arguments show only $\text{SUCCINCT-REACH}_{\alpha_q \leq k} \in \Pi_3^{\text{P}}$, where $\alpha_q$ is the $q$-independence number defined at the end of section 3.1. In particular, we would like to know the exact complexity of $\text{SUCCINCT-REACH}_{\alpha_2 \leq 1}$, i.e., the exact complexity of the reachability problem for succinctly represented graphs that become tournaments when squared.

**Acknowledgments.** We would like to thank Mitsu Ogihara, Ken Regan, Alan Selman, and Leen Torenvliet for helpful discussions. We are also grateful to the anonymous referees for pointing out mistakes in the references and for giving helpful criticism.

## REFERENCES

[1] M. Ajtai, *$\Sigma_1^1$-formulae on finite structures*, Ann. Pure Appl. Logic, 24 (1983), pp. 1–48.
[2] D. A. M. Barrington, N. Immerman, and H. Straubing, *On uniformity within* NC$^1$, J. Comput. System Sci., 41 (1990), pp. 274–306.
[3] B. Borchert and A. Lozano, *Succinct circuit representations and leaf languages are basically the same concept*, Inform. Process. Lett., 58 (1996), pp. 211–215.
[4] S. A. Cook and P. McKenzie, *Problems complete for deterministic logarithmic space*, J. Algorithms, 8 (1987), pp. 385–394.
[5] A. Ehrenfeucht, *An application of games to the completeness problem for formalized theories*, Fund. Math., 49 (1961), pp. 129–141.
[6] K. Etessami, *Counting quantifiers, successor relations, and logarithmic space*, J. Comput. System Sci., 54 (1997), pp. 400–411.
[7] R. J. Faudree, R. J. Gould, L. Lesniak, and T. Lindquester, *Generalized degree conditions for graphs with bounded independence number*, J. Graph Theory, 19 (1995), pp. 397–409.
[8] R. Fraïssé, *Sur quelques classifications des systèmes de relations*, Publ. Sci. Univ. Alger. Sér. A, 1 (1954), pp. 35–182.

[9] M. Furst, J. B. Saxe, and M. Sipser, *Parity, circuits, and the polynomial-time hierarchy*, Math. Systems Theory, 17 (1984), pp. 13–27.

[10] H. Galperin and A. Wigderson, *Succinct representations of graphs*, Inform. and Control, 56 (1983), pp. 183–198.

[11] G. Gottlob, N. Leone, and H. Veith, *Succinctness as a source of complexity in logical formalisms*, Ann. Pure Appl. Logic, 97 (1999), pp. 231–260.

[12] F. Harary and L. Moser, *The theory of round robin tournaments*, Amer. Math. Monthly, 73 (1966), pp. 231–246.

[13] L. A. Hemaspaandra and L. Torenvliet, *Optimal advice*, Theoret. Comput. Sci., 154 (1996), pp. 367–377.

[14] N. Immerman, *Expressibility and parallel complexity*, SIAM J. Comput., 18 (1989), pp. 625–638.

[15] N. Immerman, *Descriptive Complexity*, Grad. Texts Comput. Sci., Springer-Verlag, New York, 1998.

[16] N. D. Jones, *Space-bounded reducibility among combinatorial problems*, J. Comput. System Sci., 11 (1975), pp. 68–85.

[17] N. D. Jones, Y. E. Lien, and W. T. Laaser, *New problems complete for nondeterministic log space*, Math. Systems Theory, 10 (1976), pp. 1–17.

[18] H. Landau, *On dominance relations and the structure of animal societies.* III. *The condition for a score structure*, Bull. Math. Biophys., 15 (1953), pp. 143–148.

[19] H. R. Lewis and C. H. Papadimitriou, *Symmetric space-bounded computation*, Theoret. Comput. Sci., 19 (1982), pp. 161–187.

[20] N. Megiddo and U. Vishkin, *On finding a minimum dominating set in a tournament*, Theoret. Comput. Sci., 61 (1988), pp. 307–316.

[21] J. W. Moon, *Topics on Tournaments*, Holt, Rinehart, and Winston, New York, Montreal, London, 1968.

[22] A. Nickelsen and T. Tantau, *On reachability in graphs with bounded independence number*, in Proceedings of the 8th Computing and Combinatorics Conference, Lecture Notes in Comput. Sci. 2387, Springer-Verlag, New York, 2002, pp. 554–563.

[23] C. H. Papadimitriou and M. Yannakakis, *A note on succinct representations of graphs*, Inform. and Control, 71 (1986), pp. 181–185.

[24] K. Reid and L. Beineke, *Tournaments*, in Selected Topics in Graph Theory, Academic Press, New York, 1978, pp. 169–204.

[25] W. J. Savitch, *Relationships between nondeterministic and deterministic tape complexities*, J. Comput. System Sci., 4 (1970), pp. 177–192.

[26] M. Schaefer, *Deciding the Vapnik-Chervonenkis dimension is $\Sigma_3^p$-complete*, J. Comput. System Sci., 58 (1999), pp. 177–182.

[27] M. Schaefer and C. M. Umans, *Completeness in the polynomial-time hierarchy: Part* II, SIGACT News, 33 (2002), pp. 22–36.

[28] T. Tantau, *Logspace optimization problems and their approximability properties*, in Proceedings of the 15th International Symposium on Fundmentals of Computation Theory, Lecture Notes in Comput. Sci., Springer-Verlag, New York, to appear.

[29] T. Tantau, *A logspace approximation scheme for the shortest path problem for graphs with bounded independence number*, in Proceedings of the 21st Symposium on Theoretical Aspects of Computer Science, Lecture Notes in Comput. Sci. 2996, Springer-Verlag, Berlin, 2004, pp. 326–337.

[30] S. Toda, *Counting Problems Computationally Equivalent to Computing the Determinant*, Tech. report CSIM 91-07, Department of Computer Science and Information Mathematics, University of Electro-Communications, Tokyo, 1991.

[31] P. Turán, *Egy gráfelméti szélsöérték feladatról*, Mat. Fiz. Lapok, 48 (1941), pp. 436–452 (in Hungarian).

[32] P. Turán, *On the theory of graphs*, Colloq. Math., 3 (1954), pp. 19–30.

[33] C. M. Umans, *Approximability and Completeness in the Polynomial Hierarchy*, Ph.D. thesis, University of California, Berkeley, CA, 2000.

[34] H. Veith, *Succinct representation, leaf languages, and projection reductions*, Inform. and Comput., 142 (1998), pp. 207–236.

[35] K. W. Wagner, *The complexity of problems concerning graphs with regularities*, in Proceedings of the 7th Symposium on Mathematical Foundations of Computer Science, Lecture Notes in Comput. Sci. 176, Springer-Verlag, Berlin, 1984, pp. 544–552.

[36] K. W. Wagner, *The complexity of combinatorial problems with succinct input representation*, Acta Inform., 23 (1986), pp. 325–356.

[37] C. Wrathall, *Complete sets and the polynomial-time hierarchy*, Theoret. Comput. Sci., 3 (1976), pp. 23–33.

SIAM J. COMPUT. © 2005 Society for Industrial and Applied Mathematics
Vol. 34, No. 5, pp. 1196–1215

# SEPARATING THE POWER OF MONOTONE SPAN PROGRAMS OVER DIFFERENT FIELDS[*]

AMOS BEIMEL[†] AND ENAV WEINREB[†]

**Abstract.** Monotone span programs represent a linear-algebraic model of computation. They are equivalent to linear secret sharing schemes and have various applications in cryptography and complexity. A fundamental question regarding them is how the choice of the field in which the algebraic operations are performed affects the power of the span program. In this paper we prove that the power of monotone span programs over finite fields of different characteristics is incomparable; we show a superpolynomial separation between any two fields with different characteristics, solving an open problem of Pudlák and Sgall [*Algebraic models of computation and interpolation for algebraic proof systems*, in Proof Complexity and Feasible Arithmetic, DIMACS Ser. Discrete Math. Theoret. Comput. Sci. 39, P. W. Beame and S. Buss, eds., AMS, Providence, RI, 1998, pp. 279–296]. Using this result we prove a superpolynomial lower bound for monotone span programs for a function in uniform-$\mathcal{NC}^2$ (and therefore in $\mathcal{P}$), solving an open problem of Babai, Gál, and Wigderson [*Combinatorica*, 19 (1999), pp. 301–319]. (All previous superpolynomial lower bounds for monotone span programs were for functions not known to be in $\mathcal{P}$.) Finally, we show that quasi-linear secret sharing schemes, a generalization of linear secret sharing schemes introduced in Beimel and Ishai [*On the power of nonlinear secret-sharing*, in Proceedings of the 16th Annual IEEE Conference on Computational Complexity, 2001, pp. 188–202], are stronger than linear secret sharing schemes. In particular, this proves, without any assumptions, that nonlinear secret sharing schemes are more efficient than linear secret sharing schemes.

**Key words.** monotone span programs, algebraic models of computation, lower bounds, secret sharing

**AMS subject classifications.** 68Q05, 68Q17, 68R05, 68Q70

**DOI.** 10.1137/S0097539704444038

## 1. Introduction.

The relation between computational complexity and linear algebra is an important research direction with two main avenues. On one hand, algebraic techniques were used to prove lower bounds in combinatorics [1, 15, 18] and complexity; see, e.g., [21, 25, 28]. On the other hand, algebraic computational models, which capture the essence of linear algebra, were defined. Such models include, for example, arithmetic circuits, Boolean circuits with $MOD_p$ gates, and the Blum–Shub–Smale model of computation [9].

In this paper we discuss the algebraic computational model of span programs, introduced by Karchmer and Wigderson [19]. Intuitively, span programs capture the power of basic linear algebraic operations—the rank and dependency of a set of vectors. More specifically, a monotone span program is presented as a matrix over some field, with rows labeled by variables. The span program accepts an input if the rows whose variables are satisfied by the input span a fixed nonzero vector. The size of a span program is its number of rows. A detailed definition is given in section 2.

This paper deals with the role of the field in algebraic models of computation. Part of the specification of algebraic models of computation, in particular span programs, is the field in which the arithmetic operations are performed. A fundamental question is how the choice of the field, and especially its characteristic, affects the power of

[*]Received by the editors May 24, 2004; accepted for publication (in revised form) February 25, 2005, published electronically June 16, 2005. A preliminary version of this paper appeared in [5].
http://www.siam.org/journals/sicomp/34-5/44403.html

[†]Department of Computer Science, Ben-Gurion University, Beer-Sheva 84105, Israel (beimel@cs.bgu.ac.il, weinrebe@cs.bgu.ac.il). The work of the second author was partially supported by a Kreitman Foundation Fellowship and by the Frankel Center for Computer Science.

the model. As different fields may differ substantially in their structures, especially when the characteristics of the fields are different, it would be natural to expect computational models defined over different fields to differ significantly in their power. A major result separating the power of algebraic models of computation over fields was the seminal paper by Smolensky for bounded depth circuits with $MOD_p$ gates [28]. Lower bounds related to the characteristic of the field are also known for polynomial calculus proofs [6]. However, the power of the field in algebraic models of computation is yet to be fully understood.

**Our results.** The main contribution of this paper is showing that the power of monotone span programs over finite fields of different characteristic is incomparable. Prior to this work, the best separation known for monotone span programs was a logarithmic separation for the threshold function [19].[1] In this paper we show a superpolynomial separation between any two fields with different characteristics, solving an open problem of [24]. That is, for every fixed prime number $p$, we describe a function, which has a small monotone span program over the field with $p$ elements, but requires a monotone span program of size $n^{\Omega(\sqrt{\log n})}$ over any field whose characteristic is not $p$ (including fields with characteristic 0).

Our second contribution concerns the functions for which lower bounds for monotone span programs have been proved. The best known lower bound for monotone span programs, proved by Gál [13], is $n^{\Omega(\log n)}$ (improving previous results of [3, 2]). However, all the known superpolynomial lower bounds [2, 13, 14] were for functions in $\mathcal{NP}$, not known to be in $\mathcal{P}$. We show a lower bound of $n^{\Omega(\sqrt{\log n})}$ for a function in uniform-$\mathcal{NC}^2$ (and therefore in $\mathcal{P}$), thus solving an open problem of [2].[2]

Our third contribution concerns secret sharing schemes, which are an important tool in cryptography, introduced by Blakley [8], Shamir [26], and Ito, Saito, and Nishizeki [16, 17]. A *secret sharing scheme* enables a dealer to share a secret among a set of parties, such that only some predefined authorized subsets will be able to reconstruct the secret from their shares. The authorized sets correspond to a monotone Boolean function $f : \{0,1\}^n \to \{0,1\}$, where $n$ is the number of parties and the authorized subsets are the subsets with their characteristic vectors in $f^{-1}(1)$. The efficiency of a secret sharing scheme is the overall size of the shares given to the parties. Monotone span programs are equivalent to a subclass of secret sharing schemes called *linear secret sharing schemes.* Monotone span programs were also used in other cryptographic applications; see, e.g., [23, 11]. Beimel and Ishai [4] showed functions that, under plausible assumptions, have no efficient linear secret sharing scheme but yet have an efficient nonlinear secret sharing scheme. Furthermore, they introduced the class of quasi-linear secret sharing schemes. In this paper we show that quasi-linear secret sharing schemes are stronger than linear schemes. In particular, this proves, without any assumptions, that nonlinear schemes are more efficient than linear schemes.

**Highlights of the techniques.** Proving a separation between the power of two models of computation requires a function with both a lower bound for one model and an upper bound for the other. To get the lower bound for monotone span programs over a certain field, we use the method of [13], which is based on [25]. In the center of Gál's method is a matrix whose rank over the field is much larger than

[1]It was known that span programs over finite fields with the same characteristics basically have the same power.

[2]We note that every function which has a polynomial size monotone $\mathcal{NC}^1$ circuit has a polynomial size monotone span program, and every function which has a polynomial size span program over a small field has a polynomial size $\mathcal{NC}^2$ circuit.

its combinatorial cover number. To get the upper bound for the same function for monotone span programs over another field, we require the cover to have an additional property, which is related to the characteristic of the field. As an example, for GF(2) we require that each entry of the matrix be covered by an odd number of rectangles. Our use of combinatorial covers and their properties is borrowed from communication complexity (see [20] for background on communication complexity). In particular, we use ideas similar to [12], which considered the model of counting communication complexity.

The main technical contribution of this paper is constructing such a matrix and proving that it satisfies the desired properties. In particular, the matrix we construct checks whether two linear subspaces over GF($p$) have a nontrivial intersection. Not surprisingly, the matrix reflects linear-algebraic computations over GF($p$), which are difficult to simulate over fields with characteristics different than $p$.

**Organization.** In section 2 we supply some preliminaries. In section 3 we give a general method for proving a separation between the power of monotone span programs over fields with different characteristics. In section 4 we apply this general method to achieve a separation of $n^{\Omega(\sqrt{\log n})}$ for an explicit function. Finally, in section 5, we use this separation to exhibit a monotone function in uniform-$\mathcal{NC}^2$ that has no polynomial size monotone span program, and to prove that there exist secret sharing schemes stronger than the linear secret sharing schemes.

## 2. Preliminaries.

We start with the definition of our main computational model, span programs.

DEFINITION 2.1 (span program [19]). *A* span program *over a field $F$ is a triplet $\widehat{M} = \langle M, \rho, \vec{v} \rangle$, where $M$ is a matrix over $F$, $\vec{v}$ is a nonzero row vector called the target vector (it has the same number of coordinates as the number of columns in $M$), and $\rho$ is a labeling of the rows of $M$ by literals from $\{x_1, \ldots, x_n, \overline{x}_1, \ldots, \overline{x}_n\}$ (every row is labeled by one literal, and the same literal can label many rows).*

*A span program accepts or rejects an input by the following criterion. For every input $u \in \{0,1\}^n$ define the submatrix $M_u$ of $M$ consisting of those rows whose labels are satisfied by the assignment $u$. The span program $\widehat{M}$* accepts *$u$ if and only if $\vec{v} \in \operatorname{span}(M_u)$, i.e., some linear combination of the rows of $M_u$ gives the target vector $\vec{v}$. A span program* computes *a Boolean function $f$ if it accepts exactly those inputs $u$, where $f(u) = 1$. The* size *of $\widehat{M}$ is the number of rows in $M$.*[3]

*A span program is called* monotone *if the labels of the rows are only positive literals $\{x_1, \ldots, x_n\}$. Monotone span programs compute only monotone functions, and every monotone Boolean function can be computed by a monotone span program. The size of the smallest monotone span program over $F$ that computes $f$ is denoted by* $\mathrm{mSP}_F(f)$.

*Example* 2.2. Consider the following monotone span program over GF(2):

| | | | | | |
|---|---|---|---|---|---|
| $x_2$ | 1 | 1 | 0 | 0 | 0 |
| $x_2$ | 0 | 1 | 1 | 1 | 0 |
| $x_1$ | 0 | 1 | 1 | 1 | 0 |
| $x_3$ | 0 | 1 | 0 | 1 | 1 |
| $x_4$ | 0 | 0 | 1 | 0 | 1 |

[3]The choice of the fixed nonzero vector $\vec{v}$ does not affect the size of the span program. It is always possible to replace $\vec{v}$ with another nonzero vector $\vec{v}'$ via a change of basis without changing the function computed and the size of the span program. Most often $\vec{v}$ is chosen to be the $\vec{1}$ vector (with all entries equal 1).

In this example, the target vector is $\vec{v} = \langle 1, 0, 0, 1, 1 \rangle$. There are 4 different variables labeling the rows of the matrix and the inputs are of size 4. Consider the input $\langle 0, 1, 0, 1 \rangle$. Since $x_2$ and $x_4$ are satisfied by the input, we consider the submatrix consisting of rows labeled by these variables:

$$\begin{array}{|c|c|c|c|c|c|}
\hline
x_2 & 1 & 1 & 0 & 0 & 0 \\
x_2 & 0 & 1 & 1 & 1 & 0 \\
x_4 & 0 & 0 & 1 & 0 & 1 \\
\hline
\end{array}$$

The question is whether the rows of this submatrix span the target vector $\langle 1, 0, 0, 1, 1 \rangle$. Since $\langle 1, 0, 0, 1, 1 \rangle$ is the sum, over GF(2), of the rows of the submatrix, the input is accepted by the program.

Now consider the input $\langle 1, 1, 0, 0 \rangle$. Again, we focus on the submatrix of rows labeled by $x_1$ and $x_2$, the variables satisfied by the input assignment:

$$\begin{array}{|c|c|c|c|c|c|}
\hline
x_2 & 1 & 1 & 0 & 0 & 0 \\
x_2 & 0 & 1 & 1 & 1 & 0 \\
x_1 & 0 & 1 & 1 & 1 & 0 \\
\hline
\end{array}$$

Looking at the rightmost coordinate, we see that all the submatrix entries in this column are 0, while in the target vector $\langle 1, 0, 0, 1, 1 \rangle$ the rightmost entry is 1. Hence, no linear combination of the rows of the submatrix gives the target vector. Therefore, the input is rejected by the program.

**Combinatorial rectangles and covers.** Combinatorial rectangles and covers are useful tools in communication complexity and are used in this work in a similar way. Let $X$ and $Y$ be arbitrary finite sets. A combinatorial rectangle is a set $X_0 \times Y_0$, where $X_0 \subseteq X$ and $Y_0 \subseteq Y$. A *cover* of $X \times Y$ is a set $\mathcal{R}$ of rectangles such that every pair $\langle x, y \rangle \in X \times Y$ belongs to at least one rectangle in $\mathcal{R}$.

Let $M$ be a Boolean $|X| \times |Y|$ matrix such that the rows of $M$ are indexed by the elements of $X$, and the columns of $M$ are indexed by the elements of $Y$. We say that a rectangle $R_0 = X_0 \times Y_0$, where $X_0 \subseteq X$ and $Y_0 \subseteq Y$, is a *monochromatic* rectangle if there exists a $b \in \{0, 1\}$ such that for every $x \in X_0$ and $y \in Y_0$ it holds that $M[x, y] = b$. If $b = 1$, we call $R_0$ a 1-*rectangle*, and if $b = 0$, we call $R_0$ a 0-*rectangle.* We say that a cover $\mathcal{R}$ is a *monochromatic* cover of $M$ if every rectangle $R \in \mathcal{R}$ is a monochromatic rectangle. If $\mathcal{R}$ is a set of 1-rectangles that covers all the 1-entries of $M$, then $\mathcal{R}$ is called a 1-*cover* of $M$. If $\mathcal{R}$ is a set of 0-rectangles that cover all the 0-entries of $M$, we call $\mathcal{R}$ a 0-*cover* of $M$.

**Linear subspaces.** We use basic linear algebra to find a function that is easy for span programs over one field and hard for span programs over another field. For a prime number $p$, we denote by GF$(p)$ the unique finite field with $p$ elements.

Let $k$ be a positive integer, and let $p$ be a prime. Denote by $V_k^{2k}(p)$ the set of all $k$-dimensional subspaces of GF$(p)^{2k}$, and denote by $v_k^{2k}(p)$ the number of such subspaces, that is, $v_k^{2k}(p) = \left|V_k^{2k}(p)\right|$. To prove our result, we count the number of subspaces satisfying a certain property. Toward this aim, we will use the following easy algebraic claim. We say that two linear spaces $U$ and $W$ are different if there exists a vector $\vec{v}$ such that $\vec{v} \in U$ and $\vec{v} \notin W$ or vice versa.

Claim 2.3. *Let $k$ be a positive integer, $F$ be a field, and $M$ be a matrix with $k$ rows such that* $\text{rank}_F(M) = k$. *Let $T_1, T_2$ be matrices with $k$ rows each, where $T_1 \neq T_2$. Define $M_1$ (respectively, $M_2$) to be the matrix resulting from concatenating* *the matrix $T_1$ (respectively, $T_2$) to $M$, that is, $M_i = (M|T_i)$ for $i \in \{1, 2\}$. Then, the linear spaces spanned by the rows of $M_1$ and $M_2$ are different.*

*Proof.* Since $T_1 \neq T_2$, there exists an index $j \in \{1, \ldots, k\}$, such that the rows $T_1[j]$ and $T_2[j]$ are different. Let $\vec{r} = M_1[j]$; that is, $\vec{r}$ is the $j$th row of $M_1$. We show that $\vec{r}$ is not spanned by the rows of $M_2$. Assume there exists a combination of the rows of $M_2$ that spans $\vec{r}$. That is, $\vec{r} = \sum_{i=1}^{k} \alpha_i M_2[i]$ for some $\alpha_1, \ldots, \alpha_k \in F$. Let $m$ be the number of columns in $M$, and consider the restriction of the above sum to the first $m$ coordinates. It holds that $M[j] = \sum_{i=1}^{k} \alpha_i M[i]$. Since $M$ has $k$ rows and $\operatorname{rank}_F(M) = k$, we get that $\alpha_j = 1$ and $\alpha_i = 0$ for every $i \neq j$. Thus, $\vec{r} = M_2[j]$; that is, $M_1[j] = M_2[j]$, contradicting the fact that $T_1[j] \neq T_2[j]$. $\square$

One application of Claim 2.3 is the following known corollary, which gives a lower bound on $v_k^{2k}(p)$.[4]

COROLLARY 2.4. *Let $k$ be a positive integer, and let $p$ be a prime. Then $v_k^{2k}(p) \geq p^{k^2}$.*

*Proof.* Let $I_k$ be the $k \times k$ unit matrix, $T$ be an arbitrary $k \times k$ matrix over $\mathrm{GF}(p)$, and $M_1$ be the $k \times 2k$ matrix that is a concatenation of $I_k$ and $T$. There are $p^{k^2}$ different choices of $T$, and therefore $p^{k^2}$ different ways to construct $M_1$. By Claim 2.3, each such $M_1$ represents a different element of $V_k^{2k}(p)$, and thus $v_k^{2k}(p) \geq p^{k^2}$. $\square$

It is easy to see that $v_k^{2k}(p) < p^{2k^2}$, since this is the number of ways to choose *any* $k$ vectors from $\mathrm{GF}(p)^{2k}$, and thus, we have $p^{k^2} \leq v_k^{2k}(p) < p^{2k^2}$.[5]

We will denote by $\vec{e}_j$ the $j$th unit vector, that is, the vector that is 1 in the $j$th coordinate and 0 in all the others. We say that a nonzero vector has a *leading* 1 if the first nonzero coordinate in the vector is 1. Let $p$ be a prime, $\ell$ be a positive integer, and $U$ be a subspace of dimension $\ell$ over $\mathrm{GF}(p)$. Then, the number of vectors with a leading 1 in $U$ is $\frac{p^\ell - 1}{p-1}$. We will denote by $\operatorname{char}(F)$ the characteristic of the field $F$. Finally, we denote by $[n]$ the set $\{1, \ldots, n\}$.

## 3. The general method for separation.

We want to construct a function that is hard for monotone span programs over fields with characteristic different than $p$ and easy for monotone span programs over $\mathrm{GF}(p)$, where $p$ is a prime. We use the method of [13] to get the lower bound for monotone span programs over fields with characteristic different than $p$. In the center of this method is a matrix with a large gap between its rank and the size of its monochromatic cover. To get a small upper bound for monotone span programs over $\mathrm{GF}(p)$, we shall require the cover to have an additional property, which we call 1-mod-$p$; that is, for every entry of the matrix, the number of rectangles covering it is equivalent to 1 modulo $p$. Generally speaking, the number of variables in $f$, the function for which we prove the separation, is equal to the number of rectangles in a cover. A detailed description is given below.

### 3.1. The lower bound.

Let $M$ be a matrix and $\mathcal{R}$ be a monochromatic cover of $M$. Recall that $\mathcal{R}$ is a set of rectangles. Denote $n = |\mathcal{R}|$, and $\mathcal{R} = \{R_1, \ldots, R_n\}$, where $R_i = X_i \times Y_i$. A vector in $\{0,1\}^n$ can be viewed as a characteristic vector of a subset of $\mathcal{R}$. Throughout the paper, we identify each such vector with its corresponding subset. We define two subsets of $\{0,1\}^n$, Acc and Rej. These are exactly the same sets defined by Razborov in [25], proving that a monotone function can be associated

[4]Corollary 2.4 can be proved by directly counting the elements of $V_k^{2k}$. However, since we need Claim 2.3 for other purposes, we use it to prove Corollary 2.4 as well.

[5]Actually, $v_k^{2k}(p) = O(p^{k^2})$.

with any cover. We will focus on functions that accept every $x \in \text{Acc}$ and reject every $y \in \text{Rej}$.

We first define a set Acc. For every row $x$ of $M$ we define a vector $\vec{z}_x \in \{0,1\}^n$. The $i$th coordinate in $\vec{z}_x$ indicates if the rectangle $R_i$ covers the row $x$ of $M$. That is, $\vec{z}_x[i] = 1$ if $x \in X_i$, and $\vec{z}_x[i] = 0$ otherwise. The set Acc contains the vectors $\vec{z}_x$ for every row $x$ of the matrix $M$. That is, $\text{Acc} = \{\vec{z}_x : x \in X\}$. An example for Acc is described in Figure 3.1. For example, the set in Acc corresponding to $x$ in the figure is $\{R_4, R_5, R_6\}$, the rectangles that cover the row $x$, and $\vec{z}_x = \langle 0,0,0,1,1,1\rangle$.

We now define a set Rej. For every column $y$ of $M$ we define a vector $\vec{w}_y \in \{0,1\}^n$. The $i$th coordinate of $\vec{w}_y$ indicates if the rectangle $R_i$ *does not* cover the column $y$ of the matrix $M$. That is, $\vec{w}_y[i] = 1$ if $y \notin Y_i$, and $\vec{w}_y[i] = 0$ otherwise. The set Rej contains the vectors $\vec{w}_y$ for every column $y$ of the matrix $M$. That is, $\text{Rej} = \{\vec{w}_y : y \in Y\}$. For example, the set in Rej corresponding to $y$ described in Figure 3.1 is $\{R_1, R_2, R_4\}$, the rectangles that *do not* cover the column $y$, and $\vec{w}_y = \langle 1,1,0,1,0,0\rangle$.

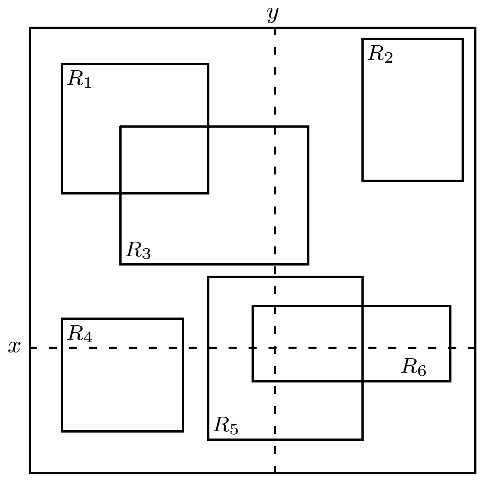


FIG. 3.1. *An illustration of elements in the sets* Acc *and* Rej. *Note that the rectangles in the figure* do not *form a cover.*

The lower bound on the size of monotone span programs is achieved using the following theorem, which is a (slightly different) restatement of Theorem 4.1 of [13].

THEOREM 3.1 (see [13]). *Let $M$ be a Boolean matrix, $\mathcal{R}$ be a monochromatic cover of $M$ of size $n$, and* Acc *and* Rej *be as defined above. If $f : \{0,1\}^n \to \{0,1\}$ is a monotone function such that $f(x) = 1$ for every $x \in$* Acc*, and $f(y) = 0$ for every $y \in$* Rej*, then* $\text{mSP}_F(f) \geq \text{rank}_F(M)$ *for every field $F$.*

That is, we get the lower bound for every function $f$ accepting Acc and rejecting Rej. Note that there are no requirements concerning inputs $t \notin (\text{Acc} \cup \text{Rej})$ (except for monotonicity). One can observe that such a function exists.

**3.2. The upper bound.** To prove a gap between the power of monotone span programs over the different fields, we need a function that has a small monotone span program over GF($p$). Toward this aim, we require the cover $\mathcal{R}$ to be a *monochromatic* 1-mod-$p$ *cover* according to the following definition.

DEFINITION 3.2. *Let $M$ be a Boolean matrix. A set $\mathcal{R}$ of combinatorial rectangles is called a* monochromatic 1-mod-$p$ cover *of $M$ if $\mathcal{R}$ is a monochromatic cover of $M$,* *and, for each entry of $M$, the number of rectangles covering it is equivalent to* 1 *modulo $p$.*

Given a small monochromatic 1-mod-$p$ cover of $M$, we construct a monotone span program over $\mathrm{GF}(p)$ that accepts Acc and rejects Rej. The gap will hold for the function computed by this span program.

Consider the following monotone span program $\widehat{P}$ over $\mathrm{GF}(p)$. The program $\widehat{P}$ associates a row with each rectangle of $\mathcal{R}$, and a column with each column of the matrix $M$. Therefore, the rectangle $R_j \in \mathcal{R}$ is represented by the variable $x_j$. The row associated with the rectangle $R_i = X_i \times Y_i$ is 1 in the column labeled $y$ if $y \in Y_i$, that is, if the rectangle $R_i$ covers the column $y$ in $M$. Otherwise, this entry in $\widehat{P}$ is 0. Note that $size(\widehat{P}) = n$; that is, there is exactly one row for each variable.

The following lemma is a simple special case of the upper bound part of Theorem 3.4 in [13]. In fact, the program $\widehat{P}$ considered here is exactly the program that one obtains by applying the construction from the proof of the upper bound in [13] to this simple special case.

LEMMA 3.3. *The program $\widehat{P}$ accepts every $\vec{z}_x \in$ Acc and rejects every $\vec{w}_y \in$ Rej.*

*Proof.* We first prove that $\widehat{P}$ accepts every $\vec{z}_x \in$ Acc. Specifically, we will show that since $\mathcal{R}$ is a 1-mod-$p$ cover, the sum of the rows labeled by the rectangles of $\vec{z}_x$ is the vector $\vec{1}$, and thus $\vec{z}_x$, is accepted by $\widehat{P}$. That is, we show that for every column of $\widehat{P}$, the rows labeled by variables satisfied by $\vec{z}_x$ sum to 1 in this column. Toward this goal, fix a column $y$. Since $\vec{z}_x \in$ Acc, it is the characteristic vector of the set of rectangles covering the row $x$ of $M$. According to the definition of $\widehat{P}$, for every rectangle $R_j$ such that $\vec{z}_x[j] = 1$, the entry $\langle R_j, y\rangle$ of $\widehat{P}$ is 1 if and only if $R_j$ covers the column $y$, that is, $y \in Y_j$. On the other hand, $\vec{z}_x[j] = 1$ if and only if $R_j$ covers the row $x$. Thus, the sum over the rows of $\widehat{P}$ associated with $\vec{z}_x$ in the column $y$ is exactly the number of rectangles covering both $y$ and $x$, that is, the number of rectangles covering the entry $\langle x, y\rangle$ in $M$. Since $\mathcal{R}$ is a 1-mod-$p$ cover, this number is 1 modulo $p$. To conclude, the sum of the rows labeled by variables that are satisfied by $\vec{z}_x$ is the vector $\vec{1}$, and $\vec{z}_x$ is accepted by $\widehat{P}$.

Let $\vec{w}_y \in$ Rej. We show that there is no linear combination of the rows labeled by the rectangles of $y$ that give the vector $\vec{1}$. Since $\vec{w}_y \in$ Rej, it is the characteristic vector of the subset of rectangles from $\mathcal{R}$ that *do not* cover the column $y$ of $M$. Hence, all the rows of $\widehat{P}$ corresponding to variables satisfied by $\vec{w}_y$ are 0 in the column associated with $y$. Therefore, every combination of the rows labeled by variables satisfied by $\vec{w}_y$ is 0 in this column. Thus, the vector $\vec{1}$ is not a linear combination of these rows, and $\vec{w}_y$ is rejected by $\widehat{P}$. ◻

Combining Theorem 3.1 and Lemma 3.3, we get the separation theorem.

THEOREM 3.4 (separation theorem). *Let $M$ be a Boolean matrix and $\mathcal{R}$ be a monochromatic* 1-mod-$p$ *cover of $M$ of size $n$. Then there exists a monotone function $f$, with $n$ variables, such that* $\mathrm{mSP}_{\mathrm{GF}(p)}(f) = n$ *and* $\mathrm{mSP}_F(f) \geq \mathrm{rank}_F(M)$ *for every field $F$.*

*Proof.* Denote by $f_{\widehat{P}}$ the function computed by $\widehat{P}$. By Lemma 3.3, $f_{\widehat{P}}$ accepts Acc and rejects Rej, and thus by Theorem 3.1 $\mathrm{mSP}_F(f_{\widehat{P}}) \geq \mathrm{rank}_F(M)$. On the other hand, $\mathrm{size}(\widehat{P}) = n$, and thus $\mathrm{mSP}_{\mathrm{GF}(p)}(f) = n$. The function $f_{\widehat{P}}$ is monotone, as it is computed by a monotone span program. ◻

## 4. The linear subspaces zero intersection function.

In this section we show an explicit matrix, with a high rank over fields with characteristic different than $p$, and a small monochromatic 1-mod-$p$ cover. Thus, by Theorem 3.4 we get a function

$f$ with a superpolynomial gap between $\mathrm{mSP}_{\mathrm{GF}(p)}(f)$ and $\mathrm{mSP}_F(f)$, where $F$ is any field such that $\mathrm{char}(F) \neq p$. We define the desired matrix in two steps: in the first step we define the matrix $M_{\mathrm{ZI}}$ and prove it has full rank over fields with $\mathrm{char}(F) \neq p$. In the second step we use $M_{\mathrm{ZI}}$ to define another matrix, $M_{\mathrm{LZI}}$, which has both a high rank over fields with char $\neq p$, and a small monochromatic 1-mod-$p$ cover.

Let $k$ be a positive integer and $p$ be a prime.[6] The *zero intersection* (ZI) function determines whether the intersection of two $k$-dimensional linear subspaces of $\mathrm{GF}(p)^{2k}$ is the subspace $\{\vec{0}\}$. More formally, define $\mathrm{ZI}_k^p : V_k^{2k}(p) \times V_k^{2k}(p) \to \{0,1\}$ as follows: $\mathrm{ZI}_k^p(U,W) = 1$, where $U$ and $W$ are subspaces in $V_k^{2k}(p)$, if and only if $\dim(U \cap W) = 0$. Recall that the intersection of any two linear subspaces is a linear subspace.

We represent $\mathrm{ZI}_k^p$ by a $v_k^{2k}(p) \times v_k^{2k}(p)$ matrix denoted $M_{\mathrm{ZI}_k^p}$. Each row and each column of $M_{\mathrm{ZI}_k^p}$ is labeled by a subspace $U \in V_k^{2k}(p)$, and each entry $M_{\mathrm{ZI}_k^p}[U,W]$ is equal to $\mathrm{ZI}_k^p(U,W)$. Denote by $r_U$ the row in $M_{\mathrm{ZI}_k^p}$ associated with the subspace $U \in V_k^{2k}(p)$. We will use ZI instead of $\mathrm{ZI}_k^p$, and $M_{\mathrm{ZI}}$ instead of $M_{\mathrm{ZI}_k^p}$, when $k$ and $p$ are clear from the context.

**4.1. Analyzing the rank of $M_{\mathrm{ZI}}$.** The next theorem shows that $M_{\mathrm{ZI}}$ has full rank over any field with char $\neq p$.

THEOREM 4.1. *Let $k$ be a positive integer, $p$ be a prime, and $F$ be a field such that* $\mathrm{char}(F) \neq p$. *Then, $M_{\mathrm{ZI}_k^p}$ has full rank over $F$.*

*Proof.* To prove that the matrix has full rank, it is sufficient to show that any unit vector is spanned by the rows of the matrix. Recall that the columns of the matrix are labeled by subspaces from $V_k^{2k}(p)$. For every $U \in V_k^{2k}(p)$ we consider the unit vector $\vec{e}_U \in \mathrm{GF}(p)^{v_k^{2k}(p)}$ and show that it is spanned by the rows of $M_{\mathrm{ZI}}$. Specifically, we show a combination of the rows of the matrix spanning $\vec{e}_U$ having a special structure: The coefficient of $\vec{r}_Z$, the row labeled $Z \in V_k^{2k}(p)$, depends only on the dimension of the subspace $U \cap Z$. More precisely, we show there are constants $\alpha_0, \dots, \alpha_k \in F$, such that

$$\vec{e}_U = \sum_{d=0}^{k} \alpha_d \sum_{\substack{Z \in V_k^{2k}(p) \\ \dim(Z \cap U) = d}} \vec{r}_Z. \tag{4.1}$$

Fix $W \in V_k^{2k}(p)$ and consider $\vec{c}_W$, the column of $M_{\mathrm{ZI}}$ associated with $W$. We have to show that with the appropriate constants $\alpha_0, \dots, \alpha_k \in F$, the above expression is 0 in this column if $W \neq U$ and is 1 if $W = U$. When computing the sum in the column $\vec{c}_W$, we add $\alpha_d$ for every subspace $Z$ such that $\mathrm{ZI}(Z,W) = 1$ (i.e., $\dim(Z \cap W) = 0$) and $\dim(Z \cap U) = d$. This motivates the following definition.

DEFINITION 4.2. *Let $U, W \in V_k^{2k}(p)$ be subspaces, and let $\ell$ be an integer such that* $\dim(U \cap W) = \ell$. *Define $H_k^p(\ell, d)$ to be the* number *of subspaces $Z \in V_k^{2k}(p)$ such that* $\dim(U \cap Z) = d$ *and* $\dim(W \cap Z) = 0$.

From symmetry arguments, the number $H_k^p(\ell, d)$ is independent of the choice of $U$ and $W$. We will write $H_k(\ell, d)$ instead of $H_k^p(\ell, d)$ when $p$ is clear from the context.

To summarize, we need to show that there are constants $\alpha_0, \dots, \alpha_k \in F$ such that the following hold:

1. For each $0 \leq \ell \leq k-1$, it holds that $\sum_{d=0}^{k} \alpha_d \cdot H_k(\ell, d) = 0$. That is, the sum over any column labeled with $W \neq U$ equals 0, where for a subspace $W \in V_k^{2k}(p)$ such that $\dim(U \cap W) = \ell$, the relevant equation is the $\ell$th equation.

[6]Throughout this section the reader should think of $k$ as small. That is, we construct a function with $n$ variables and $k \approx \sqrt{\log n}$.

2. $\sum_{d=0}^{k} \alpha_d \cdot H_k(k,d) = 1$. That is, the sum over the column associated with $U$ is 1.

Putting things differently, we view the numbers $H_k(\ell,d)$ for $\ell, d \in \{0,\dots,k\}$ as a $(k+1)\times(k+1)$ matrix over $F$.[7] According to the above conditions we have to prove there are $\alpha_0,\dots,\alpha_k \in F$ such that $H_k\langle\alpha_0,\alpha_1,\dots,\alpha_k\rangle^T = \langle 0,\dots,0,1\rangle^T$. We show that $H_k$ is invertible over $F$, and thus we can find $\alpha_0,\dots,\alpha_k$ using $H_k^{-1}$ as follows: $\langle\alpha_0,\alpha_1,\dots,\alpha_k\rangle^T = H_k^{-1}\langle 0,\dots,0,1\rangle^T$.

In the next two claims, we show that $H_k$ is upper-left triangular, where the numbers on the secondary diagonal are nonzero in $F$, and thus $H_k$ has full rank over $F$. The structure of $H_k$ is illustrated in Figure 4.1. In Claim 4.3 we show that the numbers below the secondary diagonal are all 0. In Claim 4.4 we show that the numbers on the secondary diagonal are all powers of $p$, which are nonzero since $\mathrm{char}(F) \neq p$. The numbers above the secondary diagonal may take any value from $F$.

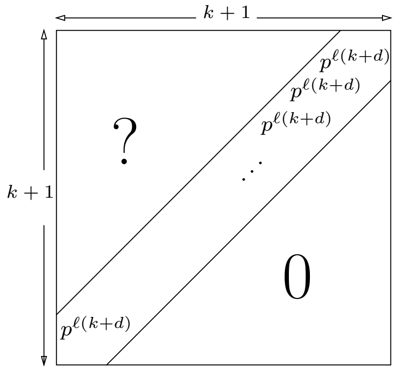


FIG. 4.1. *The structure of the matrix $H_k$.*

CLAIM 4.3. *Let $k$ be a positive integer, $\ell$ and $d$ be nonnegative integers, $p$ be a prime, and $H_k$ be as above. If $\ell + d > k$, then $H_k^p(\ell,d) = 0$.*

*Proof.* Let $U, W \in V_k^{2k}(p)$, where $\dim(U \cap W) = \ell$. We have to show that since $\ell + d > k$, there is no subspace $Z \in V_k^{2k}(p)$ such that $\dim(Z \cap U) = d$ and $\dim(Z \cap W) = 0$. Assume toward contradiction that there exists such $Z$. Let $B_{U\cap W} = \langle \vec{w}_1,\dots,\vec{w}_\ell\rangle$ be a basis of the subspace $U \cap W$. Let $B_{U\cap Z} = \langle \vec{z}_1,\dots,\vec{z}_d\rangle$ be a basis for $U \cap Z$. Consider the set of vectors $X = B_{U\cap W} \cup B_{U\cap Z}$. First note that $X \subseteq U$; that is, all the vectors in $X$ are in the subspace $U$. Since $\dim(U) = k$ and $|X| = \ell + d > k$, the set $X$ must be linearly dependent. Thus, there must be a nontrivial combination of the vectors of $X$, giving the vector $\vec{0}$, that is, $\sum_{i=1}^{\ell}\lambda_i\vec{w}_i + \sum_{i=1}^{d}\delta_i\vec{z}_i = \vec{0}$. Since both $B_{U\cap W}$ and $B_{U\cap Z}$ are linearly independent, the nonzero vector $\vec{v} = \sum_{i=1}^{\ell}\lambda_i\vec{w}_i$ is spanned by both $B_{U\cap W}$ and $B_{U\cap Z}$. Since $U \cap W \subseteq W$ and $U \cap Z \subseteq Z$, we get that $\vec{v} \in W \cap Z$, and thus $\dim(W \cap Z) > 0$, contradicting the assumption that $\dim(W \cap Z) = 0$. ▯

We need the following notation for the next claim: Let $B = \langle \vec{v}_1,\dots,\vec{v}_{2k}\rangle$ be a basis of $\mathrm{GF}(p)^{2k}$. Let $Z \in V_k^{2k}(p)$ and $B_Z = \langle \vec{z}_1,\dots,\vec{z}_k\rangle$ be a basis for $Z$. Thus there must be unique constants such that for every $i \in [k]$ we have $z_i = \sum_{j=1}^{2k}\beta_{i,j}\vec{v}_j$. Then we call the $k \times 2k$ matrix $(\beta_{i,j})$ the *representation matrix* of $B_Z$ *according to* $B$. Notice that for every basis $B$ of $Z$ we get a different representation.

[7]Since $H_k(\ell,d)$ may be a number not in $F$, we will replace it with $H_k(\ell,d) \bmod c$, where $c$ is the characteristic of $F$. If the characteristic of $F$ is 0, $H_k(\ell,d)$ will always be in $F$.

CLAIM 4.4. *Let $k$ be a positive integer, $\ell$ and $d$ be nonnegative integers, and $p$ be a prime. If $\ell + d = k$, then $H_k^p(\ell, d) = p^{\ell(k+d)}$.*

*Proof.* Let $U, W \in V_k^{2k}(p)$ be any subspaces such that $\dim(U \cap W) = \ell$. We must show that the number of subspaces $Z$ such that $\dim(Z \cap U) = d$ and $\dim(Z \cap W) = 0$ is $p^{\ell(k+d)}$. We will first define the term *canonical representation* of a subspace in $V_k^{2k}(p)$. Next, we will show that each subspace $Z$, such that $\dim(Z \cap U) = d$ and $\dim(Z \cap W) = 0$, has a canonical representation. Then we will show that every canonical representation is associated with a unique subspace $Z$ such that $\dim(Z \cap U) = d$ and $\dim(Z \cap W) = 0$. Thus, the number of such subspaces is equal to the number of different canonical representations. To complete the proof, we will show that the number of such canonical representations is $p^{\ell(k+d)}$. The canonical representation is defined according to a specific basis of $\mathrm{GF}(p)^{2k}$. Consider a basis $B_{U,W}$ of $\mathrm{GF}(p)^{2k}$ defined as follows:

$$B_{U,W} = \langle \vec{v}_1, \dots, \vec{v}_\ell, \vec{u}_1, \dots, \vec{u}_d, \vec{w}_1, \dots, \vec{w}_d, \vec{x}_1, \dots, \vec{x}_\ell \rangle,$$

where

(i) $\langle \vec{v}_1, \dots, \vec{v}_\ell \rangle$ is a basis of $U \cap W$. Recall that $\dim(U \cap W) = \ell$.

(ii) $\langle \vec{u}_1, \dots, \vec{u}_d \rangle$ is an expansion of $\langle \vec{v}_1, \dots, \vec{v}_\ell \rangle$ to a basis of $U$. Recall that $\dim(U) = k$ and $d + \ell = k$.

(iii) $\langle \vec{w}_1, \dots, \vec{w}_d \rangle$ is an expansion of $\langle \vec{v}_1, \dots, \vec{v}_\ell \rangle$ to a basis of $W$. Recall that $\dim(W) = k$ as well.

(iv) $\langle \vec{x}_1, \dots, \vec{x}_\ell \rangle$ is an expansion of $\langle \vec{v}_1, \dots, \vec{v}_\ell, \vec{u}_1, \dots, \vec{u}_{k-\ell}, \vec{w}_1, \dots, \vec{w}_{k-\ell} \rangle$ to a basis of $\mathrm{GF}(2)^{2k}$. Here there are $\ell$ vectors since $2k - (\ell + d + d) = \ell$.

We say that a subspace $Z \in V_k^{2k}(p)$ has a *canonical representation* according to $B_{U,W}$ if it has a basis whose representation matrix according to $B_{U,W}$ is as described in Figure 4.2. The matrix in Figure 4.2 is a $k \times 2k$ matrix. Each entry in zones $(b)$, $(g)$, and $(h)$ must be 0. The entries in zones $(d)$ and $(f)$ must form the unit matrices $I_\ell$ and $I_d$, respectively. Each entry in zones $(a)$, $(c)$, and $(e)$ can take any value from $\mathrm{GF}(p)$.

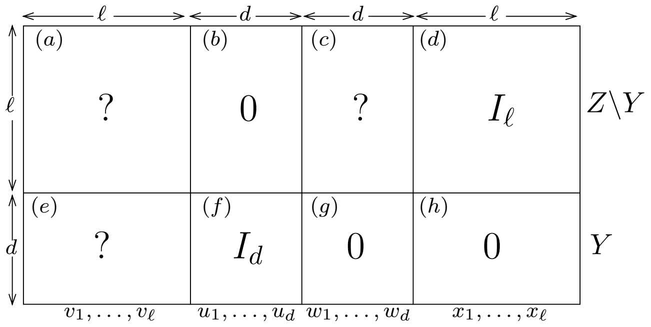


FIG. 4.2. *A canonical representation of a subspace $Z \in V_k^{2k}(p)$ with $\dim(U \cap Z) = d$ and $\dim(W \cap Z) = 0$.*

First, we show that every subspace $Z \in V_k^{2k}(p)$ such that $\dim(Z \cap U) = d$ and $\dim(Z \cap W) = 0$ has a canonical representation according to $B_{U,W}$. Let $Y = Z \cap U$. Note that $\dim(Y) = d$. Let $B_Y = \langle \vec{y}_1, \dots, \vec{y}_d \rangle$ be a basis of $Y$, and let $B_Z = \langle \vec{y}_1, \dots, \vec{y}_d, \vec{z}_1, \dots, \vec{z}_\ell \rangle$ be an expansion of $B_Y$ to a basis of $Z$. Consider $M_Z$, the representation matrix of $B_Z$ according to $B_{U,W}$. Since $Y \subseteq U$, all the entries in the zones $(g)$ and $(h)$ are 0 as required. We claim that we can perform elementary operations on the lower part of $M_Z$ so that we get the matrix $I_d$ in zone $(f)$. Otherwise, we would get a row $\vec{r}$ that is $\vec{0}$ in zone $(f)$, but this would leave all the nonzero entries of $\vec{r}$ in zone $(e)$. Since zone $(e)$ represents the basis vectors from $U \cap W$, this would mean $\dim(Z \cap W) > 0$, contradicting the properties of $Z$. It is left to set zone $(d)$ to $I_\ell$ and all the entries in zone $(b)$ to 0. Setting all the entries in zone $(b)$ to 0 can be done by elementary operations on the upper part of $M_Z$ using the rows from the lower part, which now form the unit matrix $I_d$ in zone $(f)$. (This would change the entries in zone $(a)$, but we have no constraints on this zone.) We claim that we can set zone $(d)$ to be $I_\ell$ by elementary operations on the upper part of $M_Z$. Otherwise we would get a row $\vec{r}$ that is all zero in zone $(d)$. Thus $\vec{r}$ has nonzero entries only in zones $(a)$ and $(c)$, but then it again implies that $\vec{r}$ represents a vector from $W$, contradicting the fact that $\dim(Z \cap W) = 0$.

Next we prove that *every* subspace $Z \in V_k^{2k}(p)$, which can be represented in the above canonical form, satisfies $\dim(Z \cap W) = 0$ and $\dim(Z \cap U) = d$. Let $M_Z$ be a canonical representation of $Z$ according to $B_{U,W}$. Since $M_Z$ has $I_\ell$ and $I_d$ as submatrices, we have $\text{rank}_{\text{GF}(p)}(M_Z) = k$, and thus $Z \in V_k^{2k}(p)$. Now suppose $\dim(Z \cap W) > 0$. Then we can span a vector $w \in W$ by the rows of $M_Z$. This vector has to be zero in the coordinates labeled by $\vec{u}_1, \dots, \vec{u}_d$, and by $\vec{x}_1, \dots, \vec{x}_\ell$, but this cannot be done by a nontrivial combination of the rows of $M_Z$. Thus, $\dim(W \cap Z) = 0$. The lower part of $M_Z$ is nonzero only in coordinates labeled by vectors from $U$, and since it has $I_d$ as a submatrix, we get that $\dim(Z \cap U) \geq d$. Now suppose that $\dim(Z \cap U) = d' > d$. Then we have $\dim(Z \cap U) = d'$, $\dim(Z \cap W) = 0$, and $\dim(U \cap W) = \ell$, where $\ell + d' > \ell + d = k$, which is impossible by Claim 4.3. Therefore, $\dim(U \cap Z) = d$.

To complete the proof, we show that any two subspaces having different canonical representations over $B_{U,W}$ are different. To see this, note that the matrix

$$S = \begin{pmatrix} 0 & I_\ell \\ I_d & 0 \end{pmatrix}$$

is a submatrix of any canonical representation. The matrix $S$ is clearly of rank $k$, and thus, by Claim 2.3 any two subspaces with different canonical representation are different.

Therefore, when constructing a subspace $Z$, with $\dim(Z \cap U) = d$ and $\dim(Z \cap W) = 0$, the freedom in exactly in the entries marked with "?" in Figure 4.2. Since there are $p$ possibilities for every such entry, and the number of such entries is $(k \cdot \ell) + (\ell \cdot d) = \ell(k + d)$, we conclude that $H_k(\ell, d) = p^{\ell(k+d)}$. □

Since the characteristic of $F$ is different than $p$, every power of $p$ is nonzero over $F$. Therefore, as argued above, we proved that $H_k$ has full rank over $F$, and the theorem follows. □

In Corollary 2.4 we proved that $v_k^{2k}(p) \geq p^{k^2}$. Since $M_{\text{ZI}_k}$ is a $v_k^{2k}(p) \times v_k^{2k}(p)$ matrix, $\text{rank}_F(M_{\text{ZI}_k}) \geq p^{k^2}$.

**4.2. A small 1-mod-$p$ cover for the zeros of $M_{\text{ZI}}$.** To apply Theorem 3.4 to an explicit matrix, we need this matrix to have a small monochromatic 1-mod-$p$ cover. We next show that there is a small 1-mod-$p$ cover for the 0's of $M_{\text{ZI}}$. We do not know if there exists a small 1-mod-$p$ cover for the 1's of $M_{\text{ZI}}$. Thus, we are not able to use $M_{\text{ZI}}$ directly, and we use it in section 4.3 to build the matrix $M_{\text{LZI}}$, which has a small 1-mod-$p$ cover for both the 1's and the 0's.

To gain some insight into the cover of $M_{\text{LZI}}$ we show a 1-mod-$p$ cover for the 0's of $M_{\text{ZI}}$ of size less than $p^{2k}$. This should be compared to the number of rows in

$M_{\mathrm{ZI}}$, which is $p^{\Theta(k^2)}$. Define the cover $\mathcal{R}$ as follows: Let $\vec{v} \in GF(p)^{2k}$ be a vector with a leading 1; that is, the first nonzero coordinate of $\vec{v}$ is 1. We add the rectangle $R_{\vec{v}} = X_{\vec{v}} \times Y_{\vec{v}}$ to the cover $\mathcal{R}$, where

$$X_{\vec{v}} = \left\{U \in V_k^{2k}(p) : \vec{v} \in U\right\} \text{ and } Y_{\vec{v}} = \left\{W \in V_k^{2k}(p) : \vec{v} \in W\right\}.$$

That is, $R_{\vec{v}}$ contains the rows and columns of $M_{\mathrm{ZI}}$ labeled by subspaces that contain the vector $\vec{v}$. The rectangle $R_{\vec{v}}$ is a 0-rectangle, since for each $U \in X_{\vec{v}}$ and $W \in Y_{\vec{v}}$ it holds that $\vec{v} \in U \cap W$, hence $\dim(U \cap W) \neq 0$, and thus $\mathrm{ZI}(U, W) = 0$. We claim $\mathcal{R}$ is a 1-mod-$p$ cover of the 0's of $M_{\mathrm{ZI}}$. Let $\langle U, W\rangle$ be an entry of $M_{\mathrm{ZI}}$ such that $\mathrm{ZI}(U, W) = 0$. Then $\dim(U \cap W) > 0$. Therefore, the entry $\langle U, W\rangle$ is covered by any rectangle $R_{\vec{v}}$ such that $\vec{v} \in U \cap W$, and $\vec{v}$ has a leading 1. Since $U \cap W$ is a linear subspace of $GF(p)^{2k}$, it has $\frac{p^\ell - 1}{p-1}$ vectors with a leading 1, where $\ell = \dim(U \cap W) \geq 1$. Since $\frac{p^\ell-1}{p-1} \equiv \frac{-1}{-1} \equiv 1 \pmod{p}$, the number of rectangles covering the entry $\langle U, W\rangle$ is equivalent to 1 modulo $p$. Since there are $\frac{p^{2k}-1}{p-1}$ different vectors with a leading 1 in $\mathrm{GF}(p)^{2k}$, the size of the 0-cover is $\frac{p^{2k}-1}{p-1}$.

**4.3. The list version of the zero intersection function.** To get a matrix with a high rank over fields with characteristic different than $p$, and a small monochromatic 1-mod-$p$ cover, we define the function LZI, the list version of the ZI function. The idea of using the list version of functions has been used in communication complexity [21] (see, e.g., [20]). Define $\mathrm{LZI}_k^p : (V_k^{2k}(p))^k \times (V_k^{2k}(p))^k \to \{0, 1\}$ as follows:

$$\mathrm{LZI}_k^p(\langle A_1, \ldots, A_k\rangle, \langle B_1, \ldots, B_k\rangle) = 1 \iff \exists i \in \{1 \ldots k\} \text{ such that } \mathrm{ZI}_k^p(A_i, B_i) = 1.$$

That is, $\mathrm{LZI}_k^p$ gets $k$ instances of $\mathrm{ZI}_k^p$, and outputs the value 1 if and only if $\mathrm{ZI}_k^p$ outputs 1 on at least one of the given instances. The matrix $M_{\mathrm{LZI}}$, representing LZI, is defined in a similar way to $M_{\mathrm{ZI}}$. The next two lemmas show that $M_{\mathrm{LZI}}$ has a small 1-mod-$p$ cover.

LEMMA 4.5. *There is a monochromatic* 1-mod-$p$ *cover of the* 0*'s of* $M_{\mathrm{LZI}}$ *of size smaller than* $p^{2k^2}$.

*Proof.* We build the 0-cover $\mathcal{R}_0$ of the 0's of $M_{\mathrm{LZI}}$ in a similar way to the 0-cover for $M_{\mathrm{ZI}}$ built in section 4.2. Let $\langle \vec{v}_1, \ldots, \vec{v}_k\rangle \in (\mathrm{GF}(p)^{2k})^k$ be a tuple of $k$ vectors from $\mathrm{GF}(p)^{2k}$, each with a leading 1. The rectangle in $\mathcal{R}_0$ corresponding to $\langle \vec{v}_1, \ldots, \vec{v}_k\rangle$ is $R = X \times Y$, where

$$X = \{\langle A_1, \ldots, A_k\rangle \in (V_k^{2k}(p))^k : \vec{v}_i \in A_i \text{ for each } i \in [k]\}, \text{ and}$$
$$Y = \{\langle B_1, \ldots, B_k\rangle \in (V_k^{2k}(p))^k : \vec{v}_i \in B_i \text{ for each } i \in [k]\}.$$

First we show $R$ is a 0-rectangle. If $\langle A_1, \ldots, A_k\rangle \in X$ and $\langle B_1, \ldots, B_k\rangle \in Y$, then $\vec{v}_i \in A_i \cap B_i$ for every $i \in [k]$, and thus $\mathrm{ZI}(A_i, B_i) = 0$ for every $i \in [k]$. Therefore, $\mathrm{LZI}(\langle A_1, \ldots, A_k\rangle, \langle B_1, \ldots, B_k\rangle) = 0$.

Next we show that for every 0-entry of $M_{\mathrm{LZI}}$, the number of rectangles covering it is equivalent to 1 modulo $p$. Let $\langle\langle A_1, \ldots, A_k\rangle, \langle B_1, \ldots, B_k\rangle\rangle \in (V_k^{2k}(p))^k \times (V_k^{2k}(p))^k$ such that $\mathrm{LZI}(\langle A_1, \ldots, A_k\rangle, \langle B_1, \ldots, B_k\rangle) = 0$. The entry $\langle\langle A_1, \ldots, A_k\rangle, \langle B_1, \ldots, B_k\rangle\rangle$ is covered by any rectangle associated with a tuple of $k$ nonzero vectors $\langle \vec{v}_1, \ldots, \vec{v}_k\rangle$, such that $\vec{v}_i \in A_i \cap B_i$, for every $i \in [k]$, and has a leading 1. Since $A_i \cap B_i$ is a linear subspace, the number of vectors with a leading 1 in $A_i \cap B_i$ is $\frac{p^{\ell_i}-1}{p-1}$, where $\ell_i = \dim(A_i \cap B_i) \geq 1$. Thus, the number of rectangles covering $\langle\langle A_1, \ldots, A_k\rangle, \langle B_1, \ldots, B_k\rangle\rangle$ is a product of numbers that are equivalent to 1 modulo $p$, and therefore is equivalent to 1 modulo $p$ itself.

The number of 0-rectangles in $\mathcal{R}_0$ is the number of tuples of $k$ vectors with a leading 1 from $\mathrm{GF}(p)^{2k}$, that is, $(\frac{p^{2k}-1}{p-1})^k < p^{2k^2}$. (This is much smaller than the number of rows in $M_{\mathrm{LZI}}$, which is $p^{\Omega(k^3)}$.) $\square$

Now we show a cover $\mathcal{R}_1$ for the 1's of $M_{\mathrm{LZI}}$. The natural way to do this is to associate a rectangle $R = X \times Y$ with each pair $\langle i, U\rangle$, such that $i \in [k]$, and $U \in V_k^{2k}(p)$, where

$$X = \left\{\langle A_1, \ldots, A_k\rangle \in (V_k^{2k}(p))^k : A_i = U\right\} \text{ and}$$

$$Y = \left\{\langle B_1, \ldots, B_k\rangle \in (V_k^{2k}(p))^k : \dim(U \cap B_i) = 0\right\}.$$

That is, any input pair having $ZI(A_i, B_i) = 1$ in the $i$th instance will be covered by the rectangle associated with $i$ and $A_i$.

The problem with this choice of $\mathcal{R}_1$ is that it is not a 1-mod-$p$ cover. For example, if $\langle A_1, \ldots, A_k\rangle$ and $\langle B_1, \ldots, B_k\rangle$ have exactly $p$ instances $\langle A_i, B_i\rangle$ such that $\mathrm{ZI}(A_i, B_i) = 1$, then the number of rectangles covering the entry

$$\langle\langle A_1, \ldots, A_k\rangle, \langle B_1, \ldots, B_k\rangle\rangle$$

will be equivalent to 0 modulo $p$. To solve this problem, we require $i$ to be the index of the *first* instance of ZI such that $\mathrm{ZI}(A_i, B_i) = 1$.

LEMMA 4.6. *There is a monochromatic* 1-mod-$p$ *cover for the* 1*'s of* $M_{\mathrm{LZI}}$ *of size smaller than* $p^{4k^2}$.[8]

*Proof.* Associate a rectangle $R = X \times Y$ with any pair $\langle\langle \vec{v}_1, \ldots, \vec{v}_{i-1}\rangle, U\rangle$, where $\langle \vec{v}_1, \ldots, \vec{v}_{i-1}\rangle$ is a tuple of $i-1$ vectors with a leading 1 from $\mathrm{GF}(p)^{2k}$, where $1 \le i \le k$, and $U \in V_k^{2k}(p)$ is a subspace. The sets $X$ and $Y$ are defined as follows:

$$X = \{\langle A_1, \ldots, A_k\rangle \in (V_k^{2k}(p))^k : \vec{v}_j \in A_j \text{ for each } j \in [i-1] \text{ and } A_i = U\} \text{ and}$$
$$Y = \{\langle B_1, \ldots, B_k\rangle \in (V_k^{2k}(p))^k : \vec{v}_j \in B_j \text{ for each } j \in [i-1] \text{ and } \dim(B_i \cap U) = 0\}.$$

To see that $R$ is a 1-rectangle, take $\langle A_1, \ldots, A_k\rangle \in X$ and $\langle B_1, \ldots, B_k\rangle \in Y$. Then, $\dim(A_i \cap B_i) = \dim(U \cap B_i) = 0$, and thus $\mathrm{ZI}(A_i, B_i) = 1$. Therefore, $\mathrm{LZI}(\langle A_1, \ldots, A_k\rangle, \langle B_1, \ldots, B_k\rangle) = 1$.

We next show that for every 1-entry of $M_{\mathrm{LZI}}$, the number of rectangles covering it is equivalent to 1 modulo $p$. Let $\langle\langle A_1, \ldots, A_k\rangle, \langle B_1, \ldots, B_k\rangle\rangle \in (V_k^{2k}(p))^k \times (V_k^{2k}(p))^k$ such that $\mathrm{LZI}(\langle A_1, \ldots, A_k\rangle, \langle B_1, \ldots, B_k\rangle) = 1$. Let $i$ be the smallest index such that $\dim(A_i \cap B_i) = 0$. Then the entry $\langle\langle A_1, \ldots, A_k\rangle, \langle B_1, \ldots, B_k\rangle\rangle$ is covered by a rectangle if and only if it is associated with a pair $\langle\langle \vec{v}_1, \ldots, \vec{v}_{i-1}\rangle, A_i\rangle$ such that $\vec{v}_j \in A_j \cap B_j$ for every $j \in \{1, \ldots, i-1\}$. Since the number of vectors with a leading 1 in $A_j \cap B_j$ for every $j \in [i]$ is equivalent to 1 modulo $p$, the number of such rectangles is equivalent to 1 modulo $p$ as well.

The size of $\mathcal{R}_1$ is smaller than the number of ways to choose $k$ vectors with a leading 1 from $\mathrm{GF}(p)^{2k}$, and a subspace from $V_k^{2k}(p)$, and thus is smaller than $p^{2k^2} \cdot v_k^{2k}(p) < p^{4k^2}$. $\square$

By taking the union of the 0-cover from Lemma 4.5 and the 1-cover from Lemma 4.6, we get the following corollary.

[8]It may seem that this number is too big, but this should be compared to the dimensions of $M_{LZI}$, which is $p^{\Omega(k^3)}$.

COROLLARY 4.7. *$M_{\text{LZI}}$ has a monochromatic* 1-mod-*p cover of size smaller than* $p^{5k^2}$.

We proved in Theorem 4.1 that $\operatorname{rank}_F(M_{\text{ZI}_k}) \geq p^{k^2}$. We use this to analyze the rank of $M_{\text{LZI}_k}$ over $F$. Let $A$ be an $m \times n$ matrix and $B$ be an $r \times s$ matrix. Then the *Kronecker product* of $A$ and $B$, denoted $A \otimes B$, is an $mr \times ns$ matrix formed by multiplying each element of $A$ by the entire matrix $B$ and putting it in the place of the element of $A$. For any field $F$, for every two matrices $A$ and $B$, it holds that $\operatorname{rank}_F(A \otimes B) = \operatorname{rank}_F(A) \operatorname{rank}_F(A)$. This property of the Kronecker product, together with the De Morgan laws, implies the following lemma.

LEMMA 4.8. *Let $k$ be a positive integer and let $p$ be a prime. Then*

$$\operatorname{rank}_F(M_{\text{LZI}_k^p}) = p^{\Omega(k^3)}.$$

We are ready to prove our main result.

THEOREM 4.9 (main result). *Let $p$ be a fixed prime. Then there exists a family of functions $\{f_n\}_{n\in\mathcal{N}}$ such that $\text{mSP}_{\text{GF}(p)}(f_n) = n$ and, for every field $F$ with characteristic different than $p$, it holds that $\text{mSP}_F(f_n) = n^{\Omega(\sqrt{\log n})}$.*

*Proof.* For a positive number $k$, denote by $n_k$ the size of the monochromatic 1-mod-$p$ cover for $M_{\text{LZI}}$ given by Corollary 4.7. We first show $f_n$ for each $n$ of the form $n = n_k$ for some positive $k$. According to Corollary 4.7, $M_{\text{LZI}_k}$ has a monochromatic 1-mod-$p$ cover of size $n$, which is smaller than $p^{5k^2}$. According to Lemma 4.8, we have that $\operatorname{rank}_F(M_{\text{LZI}_k}) = p^{\Omega(k^3)}$. In terms of $n$, we have

$$n^{\sqrt{\log_p n}} \leq (p^{5k^2})^{\sqrt{\log_p (p^{5k^2})}} = (p^{5k^2})^{\sqrt{5k^2}}.$$

By Theorem 3.4, there is a function $f_n$ in $n$ variables such that $\text{mSP}_{\text{GF}(p)}(f) = n$ and $\text{mSP}_F(f) \geq p^{\Omega(k^3)} = n^{\Omega(\sqrt{\log n})}$. The last equality holds since $p$ is a constant. By padding arguments, we find that the result holds for every value of $n$. ◻

## 5. A superpolynomial lower bound for a function in uniform-$\mathcal{NC}^2$.

In this section we show a monotone function that is computable by uniform-$\mathcal{NC}^2$ circuits and does not have a polynomial size monotone span program over any field.[9] For comparison, all the previous superpolynomial lower bounds were proved for functions not known to be in $\mathcal{P}$.

Denote by $f^2 = \{f_n^2\}_{n\in\mathcal{N}}$ and $f^3 = \{f_n^3\}_{n\in\mathcal{N}}$ the families of functions given by Theorem 4.9 for $p = 2$ and $p = 3$, respectively. Define the family of functions $f = \{f_{2n}\}_{n\in\mathcal{N}}$ as $f_{2n}(x_1, \ldots, x_n, y_1, \ldots, y_n) = f_n^2(x_1, \ldots, x_n) \wedge f_n^3(y_1, \ldots, y_n)$.

We show a uniform-$\mathcal{NC}^2$ family of circuits for $f$. Let $\widehat{P_2}$ be the monotone span program over GF(2) that computes $f^2$. Recall that $\text{size}(\widehat{P_2}) = n$. As mentioned in section 2, we can assume w.l.o.g. that the number of columns in $\widehat{P_2}$ is not larger than the number of rows, which is $n$. Therefore, since linear algebra over fixed finite fields is in log-space uniform-$\mathcal{NC}^2$ [7, 22, 10, 19], there exists an $\mathcal{NC}^2$ circuit $C_2$ that computes $f^2$. Similarly, there exists an $\mathcal{NC}^2$ circuit $C_3$ that computes $f^3$. Thus, the $\mathcal{NC}^2$ circuit $C = C_2 \wedge C_3$ computes $f$.

The problem with the circuit $C$, as described, is that it is not uniform. The mere *existence* of a monotone span program with a small number of columns does not yield a uniform-$\mathcal{NC}^2$ circuit. To get uniform circuits we have to show an *explicit*

[9]In this paper, *uniform* means log-space uniform.

monotone span program with a small number of columns that can be generated in space $O(\log n)$. We do this in section 5.1.

We next show that $f$ has no small monotone span program over any field. Assume there is a polynomial size monotone span program $\widehat{Q}$ that computes $f$ over some field $F$. Let $c$ be the characteristic of $F$. If $c \neq 2$, then the restriction of $\widehat{Q}$ to inputs of the form $x_1, \ldots, x_n \cdot 1^n$ gives a new monotone span program $\widehat{Q}_2$ of polynomial size over $F$ that computes $f^2$ (as any restriction of a function with a small monotone span program has a small monotone span program [19]), contradicting the fact that $f^2$ has no polynomial size monotone span program over fields with characteristic different than 2. If $c = 2$, then $c \neq 3$ and we get the contradiction for $f^3$ in a similar way. Thus, we have the following theorem.

THEOREM 5.1. *There exists a family of monotone functions* $\{f_n\}_{n\in\mathcal{N}}$ *that is computable by a uniform-*$\mathcal{NC}^2$ *family of circuits having* $\mathrm{mSP}_F(f_n) = n^{\Omega(\sqrt{\log n})}$ *for every field* $F$.

**5.1. Reducing the number of columns.** In Theorem 4.9 we introduced a function $f_{\widehat{P}}$ such that $\mathrm{mSP}_{\mathrm{GF}(p)}(f_n) = n$ and $\mathrm{mSP}_F(f_n) = n^{\Omega(\sqrt{\log n})}$. In this section we want to construct a family of uniform-$\mathcal{NC}^2$ circuits for a function that accepts Acc and rejects Rej.

It is known that any function that has a polynomial size monotone span program has a family of $\mathcal{NC}^2$ circuits. Since any monotone span program with $m$ rows that computes a function $f$ has an equivalent monotone span program with no more than $m$ columns, we can deduce the existence of a family of $\mathcal{NC}^2$ circuits that computes $f$. However, we want a uniform family of circuits. Since any transformation from a monotone span program with an arbitrary number of columns to an equivalent program with a smaller number of columns has to go over all the columns of the large original program, we cannot use the generic span program for $f_{\widehat{P}}$, as presented in section 3.4. In this section we show a monotone span program, with a linear number of both rows and columns, that accepts Acc and rejects Rej. We show that the span program can be generated in space $O(\log n)$, and we ensure the uniformity of the $\mathcal{NC}^2$ circuits.

Let $\mathcal{R}_{\mathrm{LZI}}$ be the monochromatic 1-mod-$p$ cover of $M_{\mathrm{LZI}}$ described in Corollary 4.7, and consider the following monotone span program $\widehat{S}$:[10] The program $\widehat{S}$ has a column for each $k$-tuple $\langle \vec{v}_1, \ldots, \vec{v}_k \rangle \in (\mathrm{GF}(p)^{2k})^k$, where each $\vec{v}_i$ is a vector with a leading 1 from $\mathrm{GF}(p)^{2k}$. Thus, the number of columns in $\widehat{S}$ is smaller than the number of rectangles in $\mathcal{R}_{\mathrm{LZI}}$, and hence is linear in the number of variables. Intuitively, the columns of $\widehat{S}$ form a basis of the columns of the program $\widehat{P}$ from section 4.

Recall that in $\mathcal{R}_{\mathrm{LZI}}$ there are two types of rectangles as follows:

- **0-rectangles.** We associated a 0-rectangle with every $k$-tuple of vectors $\langle \vec{v}_1, \ldots, \vec{v}_k \rangle \in (\mathrm{GF}(p)^{2k})^k$, each with a leading 1.
- **1-rectangles.** We associated a 1-rectangle $R = X \times Y$ with any pair $\langle \langle \vec{v}_1, \ldots, \vec{v}_{i-1} \rangle, U \rangle$ such that $\langle \vec{v}_1, \ldots, \vec{v}_{i-1} \rangle$ is a tuple of $i-1$ vectors with a leading 1 from $\mathrm{GF}(p)^{2k}$, where $1 \leq i \leq k$, and $U \in V_k^{2k}(p)$ is a subspace.

Every rectangle is assigned a row in $\widehat{S}$. Let $R$ be a rectangle in $\mathcal{R}_{\mathrm{LZI}}$, and let $c$ be a column in $\widehat{S}$ labeled with the tuple $\langle \vec{v}_1, \ldots, \vec{v}_k \rangle$. Then the value of the entry $\widehat{S}[R, c]$ is defined as follows:

[10]We do not know if the function computed by the monotone span program $\widehat{S}$ is the same as the function from Theorem 4.9.

- For a 0-rectangle $R$, let $\langle \vec{u}_1, \ldots, \vec{u}_k \rangle$ be the $k$-tuple of vectors associated with $R$. We set $\widehat{S}[R, c] = 1$ if $\vec{u}_j = \vec{v}_j$ for every $j \in [k]$. Otherwise, $\widehat{S}[R, c] = 0$.
- For a 1-rectangle $R$, let $\langle \langle \vec{u}_1, \ldots, \vec{u}_{i-1} \rangle, U_i \rangle$ be the $(i-1)$-tuple of vectors and the subspace associated with $R$. In this case set $\widehat{S}[R, c] = 1$ if $\vec{u}_j = \vec{v}_j$ for every $j \in [i-1]$ and $v_i \notin U_i$. Otherwise $\widehat{S}[R, c] = 0$.

By putting the rows corresponding to 0-rectangles in the upper part of $\widehat{S}$, the upper block of $\widehat{S}$ is in fact the unit matrix $I$. To compute an entry in the lower part of $\widehat{S}$, we need only check if a vector in $\mathrm{GF}(p)^{2k}$ belongs to a subspace, where $k = O(\sqrt{\log n})$. This can be easily done in space $O(\log n)$. Thus, $\widehat{S}$ can be generated in log-space. To construct a circuit that simulates the span program, we need a circuit that tests the rank of a matrix over $\mathrm{GF}(p)$. This can also be done in space $O(\log n)$ [7, 22, 10, 19]. We next prove that the function computed by $\widehat{S}$ can be used for obtaining our lower bounds. That is, the program $\widehat{S}$ accepts every $\vec{z}_x \in \mathrm{Acc}$ and rejects every $\vec{w}_y \in \mathrm{Rej}$. This fact is proved in the following two claims.

CLAIM 5.2. *The program $\widehat{S}$ accepts every $\vec{z}_x \in$* Acc.

*Proof.* Let $\vec{z}_x \in \mathrm{Acc}$. Throughout the proof we view the characteristic vector $\vec{z}_x$ as the set of rectangles it represents. We show that the vector $\vec{1}$ is the sum of the rows labeled by rectangles $R \in \vec{z}_x$, where the computations are done over $\mathrm{GF}(p)$.

Since $\vec{z}_x \in \mathrm{Acc}$, it is the characteristic vector of the set of all the rectangles in $\mathcal{R}_{\mathrm{LZI}}$ covering the row $x$ of $M_{\mathrm{LZI}}$. Let $\langle X_1, \ldots, X_k \rangle$ be the $k$-tuple of subspaces from $V_k^{2k}$ labeling the row $x$ in $M_{\mathrm{LZI}}$. Then the rectangles in $\vec{z}_x$ are of the following two types:

(i) 0-rectangles, labeled by $\langle \vec{x}_1, \ldots, \vec{x}_k \rangle$, where $\vec{x}_j \in X_j$ for every $j \in [k]$.

(ii) 1-rectangles, labeled by $\langle \langle \vec{x}_1, \ldots, \vec{x}_{i-1} \rangle, X_i \rangle$, where $\vec{x}_j \in X_j$ for every $j \in [i-1]$.

Let $c$ be a column in $\widehat{S}$. Assume that $c$ is labeled by $\langle \vec{v}_1, \ldots, \vec{v}_k \rangle$. We show that the sum of the rows labeled by rectangles from $\vec{z}_x$ in the column $c$ is 1. More specifically, we show that there is exactly one row labeled by $\vec{z}_x$ that is 1 in the column $c$. We consider the following two different cases:

(i) $\vec{v}_j \in X_j$ for every $j \in [k]$. We divide the rectangles in $\vec{z}_x$ into three as follows:

1. The unique 0-rectangle $R \in \vec{z}_x$ labeled by $\langle \vec{v}_1, \ldots, \vec{v}_k \rangle$. According to the definition of $\widehat{S}$, we have $\widehat{S}[R, c] = 1$.
2. Other 0-rectangles. Since the upper block of $\widehat{S}$ is the unit matrix $I$, we have $\widehat{S}[R, c] = 0$ for any such rectangle.
3. 1-rectangles. If $R$ is a 1-rectangle labeled by $\langle \langle \vec{x}_1, \ldots, \vec{x}_{i-1} \rangle, X_i \rangle$, then we have that $\widehat{S}[R, c] = 0$ since $\vec{v}_i \in X_i$.

Thus, there is exactly one rectangle $R \in \vec{z}_x$ such that $\widehat{S}[R, c] = 1$, and hence the sum of the rows labeled by rectangles from $\vec{z}_x$, in the column $c$ is 1.

(ii) If case (i) does not hold, there exists an index $\ell \in [k]$ such that $\vec{v}_j \in X_j$ for every $j \in [\ell - 1]$ and $\vec{v}_\ell \notin X_\ell$. In this case, for every 0-rectangle $R \in \vec{z}_x$, it holds that $\widehat{S}[R, c] = 0$, since for every such rectangle $\vec{x}_\ell \in X_\ell$, while $\vec{v}_\ell \notin X_\ell$, and thus $\vec{x}_\ell \neq \vec{v}_\ell$. Let $R \in \vec{z}_x$ be a 1-rectangle labeled by $\langle \langle \vec{x}_1, \ldots, \vec{x}_{i-1} \rangle, X_i \rangle$, for some $i \in [k]$, where $\vec{x}_j \in X_j$ for every $j \in [i-1]$. We have to check three cases.

*Case* I. $i < \ell$. In this case $\vec{v}_i \in X_i$, because $i \in [\ell - 1]$, and thus $\widehat{S}[R, c] = 0$.

*Case* II. $i > \ell$. In this case $\vec{x}_\ell \in X_\ell$, since $\ell \in [i-1]$. On the other hand, $\vec{v}_\ell \notin X_\ell$, and thus $\vec{v}_\ell \neq \vec{x}_\ell$, with $\ell \in [i-1]$, and so $\widehat{S}[R, c] = 0$.

*Case* III. $i = \ell$. In this case the only rectangle $R \in \vec{z}_x$ satisfying $\widehat{S}[R, c] = 1$ is the rectangle labeled by $\langle\langle\vec{v}_1, \ldots, \vec{v}_{i-1}\rangle, X_i\rangle$.

Therefore, again there is exactly one rectangle $R \in \vec{z}_x$ such that $\widehat{S}[R, c] = 1$, and the sum of the rows labeled by rectangles from $\vec{z}_x$, in the column $c$ is 1. Thus, $\widehat{S}$ accepts Acc. □

It is left to prove that $\widehat{S}$ rejects Rej. This part is a little more complicated than the generic case discussed in Lemma 3.3.

CLAIM 5.3. *The program* $\widehat{S}$ *rejects every* $\vec{w}_y \in \text{Rej}$.

*Proof.* Let $\vec{w}_y \in \text{Rej}$. Throughout the proof we view the characteristic vector $\vec{w}_y$ as the set of rectangles it represents. Then there exists a column labeled by $\langle Y_1, \ldots, Y_k\rangle$ in $M_{\text{LZI}}$, such that $\vec{w}_y$ is the set of all the rectangles in $\mathcal{R}_{\text{LZI}}$ that *do not* cover this column. The rectangles in $\vec{w}_y$, i.e., rectangles not covering the column labeled by $\langle Y_1, \ldots, Y_k\rangle$, are of the following types:

- **0-rectangles.** If $R$ is a 0-rectangle labeled by $\langle\vec{x}_1, \ldots, \vec{x}_k\rangle$ not covering the column $\langle Y_1, \ldots, Y_k\rangle$, then there exist an index $i \in [k]$ such that $\vec{x}_i \notin Y_i$.
- **1-rectangles.** If $R$ is a 1-rectangle labeled by $\langle\langle\vec{x}_1, \ldots, \vec{x}_{i-1}\rangle, X_i\rangle$ and not covering $\langle Y_1, \ldots, Y_k\rangle$, then either there exists an index $j \in [i-1]$ such that $\vec{x}_j \notin Y_j$ or $\dim(X_i \cap Y_i) > 0$.

Assume toward contradiction that the vector $\vec{1}$ is a linear combination of the rows labeled by rectangles from $\vec{w}_y$. Denote by $C_y$ the set of columns of $\widehat{S}$, labeled by a $k$-tuple of vectors $\langle\vec{y}_1, \ldots, \vec{y}_k\rangle$ such that $\vec{y}_i \in Y_i$, and $\vec{y}_i$ has a leading 1 for every $i \in [k]$. We will use the submatrix of $\widehat{S}$ defined by the rows of $\vec{w}_y$ and the columns $C_y$ to contradict the existence of the above linear combination. We claim that for every $R \in \vec{w}_y$, the sum of the entries in the row labeled by $R$, over the columns in $C_y$, is 0. See the following claim.

CLAIM 5.4. *For every* $R \in \vec{w}_y$,

$$\sum_{c \in C_y} \widehat{S}[R, c] = 0.$$

*Proof.* If $R \in \vec{w}_y$ is a 0-rectangle labeled by $\langle\vec{x}_1, \ldots, \vec{x}_k\rangle$, and $c$ is a column in $C_y$ labeled by $\langle\vec{y}_1, \ldots, \vec{y}_k\rangle$, then there is an index $i \in [k]$ such that $\vec{x}_i \notin Y_i$, and since $\vec{y}_j \in Y_j$ for every $j \in [k]$, we get that $\vec{x}_i \neq \vec{y}_i$ and thus $\widehat{S}[R, c] = 0$. Therefore, $\sum_{c \in C_y} \widehat{S}[R, c] = 0$.

If $R \in \vec{w}_y$ is a 1-rectangle labeled by $\langle\langle\vec{x}_1, \ldots, \vec{x}_{i-1}\rangle, X_i\rangle$, then either there exists an index $j \in [i-1]$ such that $\vec{x}_j \notin Y_j$ or $\dim(X_i \cap Y_i) > 0$. If the former is true, then for every column $c \in C_y$ labeled by $\langle\vec{y}_1, \ldots, \vec{y}_k\rangle$ we have $\vec{x}_j \notin Y_j$ and $\vec{y}_j \in Y_j$, and thus $\vec{x}_j \neq \vec{y}_j$. Since $j \in [i-1]$, this leads to $\widehat{S}[R, c] = 0$.

The only case left to discuss is that when $R \in \vec{w}_y$ is a 1-rectangle labeled by $\langle\langle\vec{x}_1, \ldots, \vec{x}_{i-1}\rangle, X_i\rangle$, such that $\vec{x}_j \in Y_j$ for every $j \in [i-1]$, and $\dim(X_i \cap Y_i) \neq 0$. We get that $\widehat{S}[R, c] = 1$ for every column $c \in C_y$ labeled by

$$\langle\vec{x}_1, \ldots, \vec{x}_{i-1}, \vec{y}_i, \ldots, \vec{y}_k\rangle,$$

where $\vec{y}_i \notin X_i$. The number of choices for a vector $\vec{y}_i$ with a leading 1 such that $\vec{y}_i \in Y_i$ and $\vec{y}_i \notin X_i$ is the number of vectors with a leading 1 in the set $Y_i \backslash X_i = Y_i \backslash (Y_i \cap X_i)$. Since both $Y_i$ and $Y_i \cap X_i$ are linear subspaces, the number of vectors with a leading 1 is equivalent to 1 modulo $p$ in both of them. Thus the number of choices for such $\vec{y}_i$ is equivalent to 0 modulo $p$. To get the number of columns $c \in C_y$ such that $\widehat{S}[R, c] = 1$, we have to multiply the number of ways to choose $\vec{y}_i$ by the number of ways to choose $\vec{y}_{i+1}, \dots, \vec{y}_k$, but the result is still equivalent to 0 modulo $p$. ∎

The number of columns in $C_y$ is the product of the number of vectors with a leading 1 in $Y_i$ for $i \in [k]$. Since each such number is 1 modulo $p$, the number of columns in $C_y$ is equivalent to 1 modulo $p$.

Recall that we assumed $\vec{1}$ is a linear combination of the rows corresponding to the rectangles in $\vec{w}_y$. Therefore, we can write

$$\sum_{R \in \vec{w}_y} \alpha_R \cdot \widehat{S}_R = \vec{1},$$

where for each $R \in \vec{w}_y$ the constant $\alpha_R$ is in GF($p$), and $\widehat{S}_R$ is the row in $\widehat{S}$ corresponding to $R$. We compute the sum

$$\sum_{R \in \vec{w}_y} \alpha_R \sum_{c \in C_y} \widehat{S}[R, c]$$

in two different ways. Since for every column $c$ it holds that

$$\sum_{R \in \vec{w}_y} \alpha_R \widehat{S}[R, c] = 1,$$

we get

$$\sum_{R \in \vec{w}_y} \alpha_R \sum_{c \in C_y} \widehat{S}[R, c] = \sum_{c \in C_y} \sum_{R \in \vec{w}_y} \alpha_R \widehat{S}[R, c] = \sum_{c \in C_y} 1 = |C_y| = 1 \bmod p.$$

On the other hand, according to Claim 5.4, the sum over any row $R \in \vec{w}_y$ of the entries in the columns of $C_y$ is equivalent to 0 modulo $p$, and we get that

$$\sum_{R \in \vec{w}_y} \alpha_R \sum_{c \in C_y} \widehat{S}[R, c] = \sum_{R \in \vec{w}_y} \alpha_R \cdot 0 = 0 \bmod p,$$

a contradiction. Thus $\vec{1}$ is not a linear combination of the rows of $\widehat{S}$ labeled by $\vec{w}_y$, and hence $\widehat{S}$ rejects $\vec{w}_y$. ∎

**5.2. Span programs and secret sharing schemes.** Secret sharing schemes, introduced by Blakley [8], Shamir [26], and Ito, Saito, and Nishizeki [16, 17], are a cryptographic tool allowing a dealer to share a secret between a set of parties such that only some predefined authorized subsets of parties can reconstruct the secret from their shares. The reader is referred to [27] and [29] for a more formal and detailed discussion on secret sharing schemes.

The authorized sets in a secret sharing scheme are described by a monotone Boolean function $f : \{0,1\}^n \to \{0,1\}$, where $n$ is the number of parties and the authorized subsets are the subsets with their characteristic vectors in $f^{-1}(1)$. Most of the known secret sharing schemes are linear schemes, that is, schemes in which the shares are a linear combination of the secret and some random field elements. Linear schemes are equivalent to monotone span programs, where the total size of the shares is the size of the corresponding monotone span program. Beimel and Ishai [4] showed functions that, under plausible assumptions, have no efficient linear secret sharing scheme but yet have an efficient nonlinear secret sharing scheme. However, prior to this work, no secret sharing schemes were proved more powerful than linear schemes, without any assumptions.

A quasi-linear secret sharing scheme [4] is obtained by composing linear secret sharing schemes, possibly over different fields. Beimel and Ishai [4] have shown that under the assumption that the power of monotone span programs over different fields is incomparable, quasi-linear schemes are superpolynomially stronger than linear schemes. Their proof is very similar to the proof of Theorem 5.1. That is, the functions described in Theorem 5.1 have, by definition, a small quasi-linear secret sharing scheme but cannot have a small linear scheme.

THEOREM 5.5. *There is an explicit family of functions* $\{f_n\}_{n\in\mathcal{N}}$ *such that the complexity of every linear secret sharing scheme for the family is* $n^{\Omega(\sqrt{\log n})}$*, and yet the family has a polynomial quasi-linear secret sharing scheme.*

**Acknowledgments.** We thank Yinnon Haviv for his very valuable help and Anna Gál for many helpful discussions.

## REFERENCES

[1] L. BABAI AND P. FRANKL, *Linear Algebra Methods in Combinatorics: With Applications to Geometry and Computer Science*, Manuscript, Preliminary Version 2, University of Chicago, Chicago, IL, 1992. http://www.cs.uchicago.edu/research/publications/combinatorics

[2] L. BABAI, A. GÁL, AND A. WIGDERSON, *Superpolynomial lower bounds for monotone span programs*, Combinatorica, 19 (1999), pp. 301–319.

[3] A. BEIMEL, A. GÁL, AND M. PATERSON, *Lower bounds for monotone span programs*, Comput. Complexity, 6 (1997), pp. 29–45.

[4] A. BEIMEL AND Y. ISHAI, *On the power of nonlinear secret-sharing*, in Proceedings of the 16th Annual IEEE Conference on Computational Complexity, 2001, pp. 188–202.

[5] A. BEIMEL AND E. WEINREB, *Separating the power of monotone span programs over different fields*, in Proceedings of the 44th IEEE Symposium on Foundations of Computer Science, 2003, pp. 428–437.

[6] E. BEN-SASSON AND R. IMPAGLIAZZO, *Random CNF's are hard for the polynomial calculus*, in Proceedings of the 40th IEEE Symposium on Foundations of Computer Science, 1999, pp. 415–421.

[7] S. J. BERKOWITZ, *On computing the determinant in small parallel time using a small number of processors*, Inform. Process. Lett., 18 (1984), pp. 147–150.

[8] G. R. BLAKLEY, *Safeguarding cryptographic keys*, in Proceedings of the 1979 AFIPS National Computer Conference, R. E. Merwin, ed., AFIPS Conf. Proc. 48, AFIPS Press, Arlington, VA, 1979, pp. 313–317.

[9] L. BLUM, M. SHUB, AND S. SMALE, *On a theory of computation over the real numbers: NP completeness, recursive functions, and universal machines*, Bull. Amer. Math. Soc. (N.S.), 21 (1989), pp. 1–46.

[10] G. BUNTROCK, C. DAMM, U. HERTRAMPF, AND C. MEINEL, *Structure and importance of the logspace-mod class*, Math. Systems Theory, 25 (1992), pp. 223–237.

[11] R. CRAMER, I. DAMGÅRD, AND U. MAURER, *General secure multi-party computation from any linear secret-sharing scheme*, in Advances in Cryptology–EUROCRYPT 2000, B. Preneel, ed., Lecture Notes in Comput. Sci. 1807, Springer, Berlin, 2000, pp. 316–334.

[12] C. DAMM, M. KRAUSE, C. MEINEL, AND S. WAACK, *On relations between counting communication complexity classes*, J. Comput. System Sci., 69 (2004), pp. 259–280.

[13] A. GÁL, *A characterization of span program size and improved lower bounds for monotone span programs*, in Proceedings of the 30th ACM Symposium on the Theory of Computing, 1998, pp. 429–437.

[14] A. GÁL AND P. PUDLÁK, *Monotone complexity and the rank of matrices*, Inform. Process. Lett., 87 (2003), pp. 321–326.

[15] C. GODSIL AND G. ROYLE, *Algebraic Graph Theory*, Graduate Texts in Math. 207, Springer-Verlag, New York, 2001.

[16] M. ITO, A. SAITO, AND T. NISHIZEKI, *Secret sharing schemes realizing general access structure*, Electron. Comm. Japan Part III Fund. Electron. Sci., 72 (1989), pp. 56–63.

[17] M. Ito, A. Saito, and T. Nishizeki, *Multiple assignment scheme for sharing secret*, J. Cryptology, 6 (1993), pp. 5–20.
[18] S. Jukna, *Extremal Combinatorics With Applications in Computer Science*, Texts in Theoret. Comput. Sci., Springer-Verlag, Berlin, 2001.
[19] M. Karchmer and A. Wigderson, *On span programs*, in Proceedings of the 8th Annual Structure in Complexity Theory Conference, IEEE, Los Alamitos, CA, 1993, pp. 102–111.
[20] E. Kushilevitz and N. Nisan, *Communication Complexity*, Cambridge University Press, Cambridge, UK, 1997.
[21] K. Mehlhorn and E. M. Schmidt, *Las Vegas is better than determinism in VLSI and distributed computing*, in Proceedings of the 14th ACM Symposium on the Theory of Computing, 1982, pp. 330–337.
[22] K. Mulmuley, *A fast parallel algorithm to compute the rank of a matrix over an arbitrary field*, Combinatorica, 7 (1987), pp. 101–104.
[23] M. Naor, B. Pinkas, and O. Reingold, *Distributed pseudo-random functions and KDCs*, in Advances in Cryptology—EUROCRYPT '99, J. Stern, ed., Lecture Notes in Comput. Sci. 1592, Springer, Berlin, 1999, pp. 327–346.
[24] P. Pudlák and J. Sgall, *Algebraic models of computation and interpolation for algebraic proof systems*, in Proof Complexity and Feasible Arithmetic, DIMACS Ser. Discrete Math. Theoret. Comput. Sci. 39, P. W. Beame and S. Buss, eds., AMS, Providence, RI, 1998, pp. 279–296.
[25] A. A. Razborov, *Applications of matrix methods to the theory of lower bounds in computational complexity*, Combinatorica, 10 (1990), pp. 81–93.
[26] A. Shamir, *How to share a secret*, Comm. ACM, 22 (1979), pp. 612–613.
[27] G. J. Simmons, *An introduction to shared secret and/or shared control schemes and their application*, in Contemporary Cryptology. The Science of Information Integrity, G. J. Simmons, ed., IEEE Press, New York, 1992, pp. 441–497.
[28] R. Smolensky, *Algebraic methods in the theory of lower bounds for Boolean circuit complexity*, in Proceedings of the 19th ACM Symposium on the Theory of Computing, 1987, pp. 77–82.
[29] D. R. Stinson, *An explication of secret sharing schemes*, Des. Codes Crypto., 2 (1992), pp. 357–390.

SIAM J. COMPUT.
Vol. 34, No. 5, pp. 1216–1252

# EFFICIENCY OF OBLIVIOUS VERSUS NONOBLIVIOUS SCHEDULERS FOR OPTIMISTIC, RATE-BASED FLOW CONTROL*

PANAGIOTA FATOUROU†, MARIOS MAVRONICOLAS‡, AND PAUL SPIRAKIS§

**Abstract.** Two important performance parameters of distributed, *rate-based flow control algorithms* are their *locality* and *convergence complexity.* The former is characterized by the amount of global knowledge that is available to their scheduling mechanisms, while the latter is defined as the number of *update operations* performed on rates of individual *sessions* until *max-min fairness* is reached. *Optimistic* algorithms allow any *session* to intermediately receive a rate larger than its max-min fair rate; *bottleneck* algorithms finalize the rate of a session only if it is restricted by a certain, highly congested link of the network. In this work, we present a comprehensive collection of lower and upper bounds on convergence complexity, under varying degrees of locality, for optimistic, bottleneck, rate-based flow control algorithms.

Say that an algorithm is *oblivious* if its scheduling mechanism uses no information of either the session rates or the network topology. We present a novel, combinatorial construction of a capacitated network, which we use to establish a fundamental lower bound of $\frac{dn}{4} + \frac{n}{2}$ on the convergence complexity of *any* oblivious algorithm, where $n$ is the number of sessions laid out on a network, and $d$, the *session dependency*, is a measure of topological dependencies among sessions. Moreover, we devise a novel *simulation proof* to establish that, perhaps surprisingly, the lower bound of $\frac{dn}{4} + \frac{n}{2}$ on convergence complexity still holds for any *partially oblivious* algorithm, in which the scheduling mechanism is allowed to use information about session rates, but is otherwise unaware of network topology.

On the positive side, we prove that the lower bounds for oblivious and partially oblivious algorithms are both *tight.* We do so by presenting *optimal* oblivious algorithms, which converge after $\frac{dn}{2} + \frac{n}{2}$ update operations are performed in the *worst* case. To complete the picture, we show that *linear* convergence complexity can indeed be achieved if information about both session rates and network topology is available to schedulers. We present a counterexample, *nonoblivious* algorithm, which converges within an *optimal* number of $n$ update operations.

Our results imply a surprising convergence complexity collapse of oblivious and partially oblivious algorithms, and a convergence complexity separation between (partially) oblivious and nonoblivious algorithms for optimistic, bottleneck rate-based flow control.

**Key words.** distributed algorithms; lower bounds; rate-based flow control; max-min fairness; convergence complexity; oblivious, partially oblivious, and nonoblivious algorithms; bottleneck algorithms

*Received by the editors August 4, 2003; accepted for publication (in revised form) December 7, 2004; published electronically June 16, 2005. A preliminary version of this work appeared in the Proceedings of the 16th Annual ACM Symposium on Principles of Distributed Computing (Santa Barbara, CA, 1997), ACM, New York, 1997, pp. 139–148. This work was partially supported by the ESPRIT Program of the European Union under the Long Term Research Project ALCOM-IT (contract 20244), by the IST Program of the European Union under Projects ALCOM-FT (contract IST-1999-14186), FLAGS (contract IST-2001-33116), and DELIS (contract 001907), by funds from the Program E.Π.E.A.E.K. II of the Greek Ministry of Education, by research funds at University of Cyprus, and by AT&T/NCR research funds.

http://www.siam.org/journals/sicomp/34-5/43275.html

†Department of Computer Science, University of Ioannina, P.O. Box 1186, Ioannina GR-45110, Greece (faturu@cs.uoi.gr). Part of this research was performed while this author was at the Department of Computer Engineering and Informatics, University of Patras, the Research and Academic Computer Technology Institute, the Max-Planck Institute für Informatik, and while visiting the Department of Computer Science, University of Cyprus.

‡Department of Computer Science, University of Cyprus, P.O. Box 20537, Nicosia CY-1678, Cyprus (mavronic@ucy.ac.cy). Part of this research was performed while this author was at the Department of Computer Science and Engineering, University of Connecticut, while visiting the University of Patras and Research and Academic Computer Technology Institute, and while at AT&T Labs—Research, as a visitor to the DIMACS Special Year on Networks.

§Department of Computer Engineering and Informatics, University of Patras, Rion, Patras GR-26500, Greece, and Research and Academic Computer Technology Institute, P.O. Box 1122, Patras GR-26110, Greece (spirakis@cti.gr).

**AMS subject classifications.** 68M10, 68M12, 68M14, 68Q25, 68R05, 68W15, 68W40, 90B18

**DOI.** 10.1137/S009753970343275X

## 1. Introduction.

In many modern *communication networks*, a connection between different users is established by a *session*, a virtual circuit of infinite duration that involves a fixed path between a *source* and a *destination.* The role of *flow control algorithms* is to alleviate throughput degradation, unfairness, deadlocks, and failures by preventing buffer overflow from arising in situations where the externally injected load is larger than what can be accommodated even with optimal routing (see, e.g., [2, Chapter 6] or [3, 6, 12, 14, 15, 16, 17, 22]). In particular, *rate-based* flow control algorithms adjust transmission rates of different sessions in an end-to-end manner, with the objective to optimize network utilization while still maintaining fairness between different sessions (see, e.g., [1, 2, 4, 5, 11, 13, 16, 18]).

For a range of settings including both high-speed networks and Internet applications, *max-min fairness* [1, 2, 5, 6, 7, 13, 14, 15, 16, 17, 19] has emerged as a widely accepted formulation of fairness for flow control; roughly speaking, max-min fairness requires that it be impossible for any session to receive an infinitesimally larger rate on the account of a session with a smaller or equal rate [15, 16, 17, 21]. Call *max-min fair rates* those achieving max-min fairness.

Any rate-based flow control algorithm can be classified as one of two broad classes, *conservative* and *optimistic*, according to the way in which rates of sessions are adjusted. Conservative algorithms (see, e.g., [2, Chapter 6]) do not provide for decreases to the rates of sessions; in contrast, optimistic algorithms allow decreases, so that rates may go up and down and a session can intermediately receive a rate larger than its final. Afek, Mansour, and Ostfeld [1] introduced optimistic algorithms and advocated them as fitting better than the conservative ones into real dynamic networks; indeed, in such networks, new sessions may join in, so that it is desirable to incrementally adjust rates by decreasing the rates of some of them.

A crucial component of a rate-based flow control algorithm is its *scheduler*, the abstract mechanism it uses to decide which session's rate to adjust next. Apparently, it is desirable that the scheduler does not require *global* knowledge of either the session rates or the network topology. Clearly, no-knowledge schedulers are not only more efficient in terms of communication and computation, but they also adjust more easily to dynamic changes in network topology. So, one important performance parameter of a rate-based flow control algorithm is its *locality*, measured by the amount of global knowledge required by the scheduler.

Call a scheduler that uses no information of either the session rates or the network topology an *oblivious* scheduler. On the opposite extreme, a *nonoblivious* scheduler requires full knowledge of both session rates and network topology. There is, in addition, an important middle ground between oblivious and nonoblivious schedulers: schedulers which, although unaware of network topology, do have access to session rates; call these schedulers *partially oblivious.* Clearly, a partially oblivious scheduler is superior to a nonoblivious scheduler in terms of robustness to dynamic changes in network topology, while it is surpassed by an oblivious scheduler. Afek et al. [1, sections 4 and 5] presented two interesting, partially oblivious schedulers called GlobalMinSched and LocalMinSched, respectively; these schedulers always choose for an increase the session whose rate is the *globally* and *locally* smallest, respectively.

A second crucial component of a rate-based flow control algorithm is its *terminator*, the abstract mechanism it uses to decide which sessions to terminate at each specific instant.[1] *Bottleneck* terminators [16] finalize the rate of a session, thereby terminating the session, only if its rate is restricted on some particular network link that allows for the least possible share of bandwidth to each session traversing it; call such a link a *bottleneck edge* [16].

When a session is scheduled for an increase, its rate is increased by the minimum possible share of bandwidth along its path; this may involve possible decreases to the rates of crossing sessions (see, e.g., [5, 6, 7, 18, 19]). The *convergence complexity* of a rate-based flow control algorithm is the number of individual rate adjustments performed in the worst case until the algorithm terminates and rates have reached max-min fairness (so that they do not change any further). Thus, convergence complexity models the number of rounds of communication and local computation needed for convergence to max-min fairness, so that it is another significant performance parameter of such distributed algorithms.

We measure convergence complexity in terms of a particular, simple abstraction of rate adjustments, called an *update operation*, which was introduced in [1, section 2.1]. In essence, an `update` operation atomically adjusts the rates of a group of neighboring sessions in a fair and optimistic way.[2] We note that some of the essential intricacies encountered when designing practical, distributed flow-control algorithms [4, 5, 6, 7, 18, 19, 22] include scheduling the rate adjustments, minimizing the communication, and converging to fairness quickly. Although we do not address in this work implementation issues for the model used, we feel that it captures most of these intricacies. (For a discussion of such issues for this model, see [9].)

This work presents a comprehensive collection of bounds on convergence complexity under varying degrees of locality for optimistic, bottleneck, rate-based flow control algorithms; more specifically, we show upper and lower bounds (sections 5 and 7, resp.) on convergence complexity for oblivious, partially oblivious, and nonoblivious such algorithms. The lower bounds demonstrate that achieving oblivious, or even partially oblivious, scheduling, and therefore locality, necessarily imposes a nonconstant, multiplicative overhead on convergence complexity. These are the first general lower bounds ever shown on the convergence of rate-based flow control algorithms. In addition, our algorithms span a wide spectrum of convergence complexity bounds and locality properties, while they improve significantly upon all previous optimistic algorithms with respect to the combination of these two performance measures. To establish these upper bounds, we offer several new basic properties and tools for the design and analysis of optimistic, bottleneck, rate-based flow control algorithms (section 4); these properties significantly extend and strengthen the corresponding ones shown in [1].

Our bounds are expressed in terms of $n$, the total number of sessions laid out on the network, and a new parameter $d$, called *session dependency*, through which we derive more accurate bounds on convergence complexity. Specifically, $d$ is the maximum number of sessions that share an edge either directly or indirectly. In the rest of this paper, we will focus on optimistic and bottleneck algorithms; we sometimes refer to them simply as algorithms.

[1]In all of our discussion, a terminated session is meant to be one that has reached its final (max-min fair) rate, so that its rate value will not change any further.

[2]In more detail, an `update` operation increases, if possible, the rate of one session, so that it becomes the session with the maximum rate on some particular link that allows the minimum possible increase. The new value is a function of the link capacity and the rates of other sessions traversing the link. Conflicting sessions whose rates exceed the new value have their rates decreased to the new value, while rates of other sessions remain unchanged. These adjustments saturate that particular link.

Our first major result is a fundamental lower bound on the convergence complexity of oblivious algorithms. Its proof relies on constructing, given any arbitrary oblivious algorithm, a specific network, as a function of the algorithm's scheduler, so that if sessions are scheduled on this network according to the scheduler and the algorithm computes the max-min fair rates, then $\Omega(dn)$ `update` operations are required before convergence (section 6). The construction is novel, purely combinatorial, and of independent interest. We rely on the algorithm being optimistic and bottleneck to show that convergence is slow.

Although intuition may suggest that knowledge of session rates can be crucial to performance, we surprisingly establish that the lower bound of $\Omega(dn)$ that we show on the convergence complexity of oblivious algorithms applies also to partially oblivious algorithms. We use a powerful simulation proof to simulate any partially oblivious scheduler on some network by an oblivious scheduler on the same network. The simulation inherits the $\Omega(dn)$ lower bound shown for oblivious algorithms down to partially oblivious algorithms.

We show a matching upper bound on the convergence complexity of oblivious and partially oblivious algorithms. We present a class of oblivious algorithms, called $d$-Epoch, with convergence complexity $\Theta(dn)$; an example is algorithm RoundRobin, which uses the simple and natural idea of scheduling sessions in a round-robin order. We note that the partially oblivious algorithms GlobalMin and LocalMin from [1, sections 4 and 5] also achieve convergence complexity $\Theta(dn)$, which implies the tightness of our lower bound for partially oblivious algorithms; however, any $d$-Epoch algorithm improves over these two algorithms since it is oblivious.

At this point, it is only natural to ask whether it is possible to overcome the $\Theta(dn)$ barrier on convergence complexity achievable by oblivious or partially oblivious algorithms, possibly at the cost of sacrificing locality. Perhaps not very surprisingly, it turns out that the locality enjoyed by oblivious and partially oblivious algorithms comes at a multiplicative in $d$ overhead on convergence complexity. We present a counterexample, nonoblivious and optimistic algorithm called Linear that achieves an exact bound of $n$ on convergence complexity; Linear follows the earlier idea of selecting and conservatively increasing the rate of any session that traverses the most highly congested link in the network (see, e.g., [2, section 6.4.2] or [15]). We emphasize that Linear, although theoretically efficient, is clearly impractical.

Our work differs at first from earlier work on rate-based flow control carried out in the data networks community (see, e.g., [2, 3, 11, 12, 15, 16, 17, 22]) in adopting the optimistic approach introduced in [1]. We have been able to show that certain classical scheduling policies, such as round-robin [13, 14] or scheduling a session that traverses the most congested link [15], are correctly applicable in the optimistic framework. Most interestingly, we have shown the first general lower bounds on the convergence complexity of rate-based flow control algorithms; these imply that some scheduling policies are provably superior over some other policies, in terms of either convergence complexity or locality (or both), and even optimal.

**2. Model.** Many of our definitions formalize and refine those from [1] and [2, Chapter 6]. A *communication network* is a graph $G = (V, E)$, where vertices and edges represent switches and links, respectively. A set $\mathcal{S}$ of $n \geq 1$ *sessions* is laid out on $G$. Each session $S_i$ (also denoted as $i$) passes through a sequence of edges and has a *rate* $r(S_i)$. The vector $\mathbf{r} = \langle r(S_1), \ldots, r(S_n) \rangle$ is the *rate vector*. Each edge has a *capacity* $c(e) > 0$, which the sum of the rates of the sessions traversing it cannot exceed; when this sum equals the capacity, the edge is *saturated*. In a *max-min fair rate vector*

[15, 16, 17, 21], an increase to the rate of any session requires either exceeding an edge capacity or decreasing the rate of another session with equal or lower rate.

The communication network is abstracted as a state machine. Each state $Q = \langle \mathbf{r}_Q, \mathcal{A}_Q \rangle$ consists of a *rate vector* $\mathbf{r}_Q$ and a set $\mathcal{A}_Q \subseteq \mathcal{S}$ of *active sessions.* Intuitively, an active session is one that has not yet reached its max-min fair rate. Denote by $\mathcal{D}_Q = \mathcal{S} \setminus \mathcal{A}_Q$ the set of *done sessions* in state $Q$. In the *initial state* $Q^{in}$, all sessions are active and have zero rates. A state is *final* if all sessions have reached their max-min fair rates.

An operation defines a procedure to compute new rates for a set of sessions on the basis of their old rates. Formally, an *operation* is a function `operation` that takes as input a session $S_i$ and a state $Q$, and outputs new rates for $S_i$ and all sessions $S_j$ sharing edges with $S_i$. Say that `operation` is *conservative* if it decreases no individual rate; else `operation` is *optimistic.*

We will consider a specific optimistic operation, called `update`, which we describe below. For a state $Q$ and each edge $e$ traversed by $S_i$, $\Delta_Q(i, e)$ is the maximum amount by which $r_Q(S_i)$ can be increased (without exceeding the capacity of $e$) while decreasing down to the increased rate of $S_i$ the rates of other active sessions passing through $e$ and exceeding the increased rate. Notice that these rate adjustments saturate $e$. Intuitively, $\Delta_Q(i, e)$ is the maximum amount by which $r_Q(S_i)$ can be increased in a fair manner if edge $e$ were the only edge constraining $S_i$. Finally, $S_i$'s rate is increased by $\Delta_Q(i)$, called the *increase* for $S_i$ in $Q$, which is the minimum of these maximum amounts over all edges that $S_i$ passes through. In addition, each (active) session $S_j$ sharing an edge with $S_i$ has its rate decreased down to the new rate $r(S_i)$ (unless it is already less). We remark that the rates computed by `update` saturate the edge(s) realizing $\Delta_Q(i)$, but no other edges. Notice also that the `update` operation uses only local information with respect to session $S_i$ [17, section IV]. From now on, we will consider only optimistic algorithms that employ the `update` operation.

A scheduler decides the order of sessions on which to perform the `update` operation. Formally, a *scheduler* (cf. [1]) is a function $\mathsf{Sched}$ that maps a pair $\langle G, \mathcal{S} \rangle$, a state $Q$, and a state index $l \geq 1$ to a session $i = \mathsf{Sched}(\langle G, \mathcal{S} \rangle, Q, l)$. A *terminator* is a function $\mathsf{Term}$ that maps a pair $\langle G, \mathcal{S} \rangle$ and a state $Q$ to a set of sessions $\mathcal{T} = \mathsf{Term}(\langle G, \mathcal{S} \rangle, Q) \subseteq \mathcal{A}_Q$; intuitively, $\mathsf{Term}$ decides which active sessions will be *terminated* (and their rates will not change any further). An *algorithm* is a pair $\mathsf{Alg} = \langle \mathsf{Sched}, \mathsf{Term} \rangle$. The classification of operations into conservative and optimistic leads to the corresponding classification of algorithms in a natural way.

An *oblivious* scheduler uses no knowledge of either the topology of the network or the rates and *statuses* (active or done) of sessions. Thus, an oblivious scheduler is a (finite or infinite) sequence $\mathsf{Sched} = i_1, i_2, \ldots$, where for each $l \geq 1$, $i_l \in [n]$. A *partially oblivious* scheduler uses no knowledge of the topology of the network, while it may use knowledge of the rates (and statuses) of sessions. Notice that any oblivious scheduler is also partially oblivious, but not vice versa. A *nonoblivious* scheduler is a scheduler that is not partially oblivious. The classification of schedulers into oblivious, partially oblivious, and nonoblivious leads to the corresponding classification of algorithms in a natural way.

For any edge $e \in E$ and state $Q$, the *allotted capacity* of $e$ in $Q$ [1, section 2.1], denoted $allot_Q(e)$, is the total rate already allocated to done sessions passing through the edge. Clearly, $\sum_{S \in \mathcal{A}_Q | e} r_Q(S) \leq c(e) - allot_Q(e)$, where $\mathcal{A}_Q \mid e$ denotes the set of active sessions traversing $e$ in $Q$. For any state $Q$ and for any edge $e$ with $|\mathcal{A}_Q \mid e| > 0$ active sessions, the *fair share* of $e$ in state $Q$ [1, section 2.1], denoted $FS_Q(e)$, is defined to be $FS_Q(e) = \frac{c(e) - allot_Q(e)}{|\mathcal{A}_Q \mid e|}$; intuitively, $FS_Q(e)$ is the per-session share of the portion of the capacity of edge $e$ which has not yet been allocated to sessions done in state $Q$ that traverse the edge. For any state $Q$, a *bottleneck edge* for $Q$ [16] is an edge $e$ such that for each active session passing through $e$, $FS_Q(e)$ is the minimum fair share over all edges traversed by the session.

A terminator $\mathsf{Term}$ is *bottleneck* [16] if for any state $Q$ and session $S$, $S \in \mathsf{Term}(\langle G, \mathcal{S}\rangle, Q)$ if (and only if) there exists an edge $e$ traversed by $S$ such that (1) $e$ is a bottleneck edge for $Q$, and (2) $r_Q(S) = FS_Q(e)$. Whenever such an edge $e$ exists for state $Q$ and $\mathsf{Term}$ is bottleneck, we will say that $e$ causes the termination of session $S$ in $Q$. Say that an algorithm $\mathsf{Alg}$ is *bottleneck* if $\mathsf{Term}$ is.

The *execution* $\alpha$ of $\mathsf{Alg}$ on $\langle G, \mathcal{S}\rangle$ is an infinite sequence of alternating states and session indices $\alpha = Q_0, i_1, Q_1, \ldots, i_l, Q_l, \ldots$, satisfying the following conditions:

(1) $Q_0 = Q^{in}$ and $\mathcal{A}_{Q_0} = \mathcal{A}_{Q_1} = \mathcal{S}$;
(2) for each $l \geq 1$, $i_l = \mathsf{Sched}(\langle G, \mathcal{S}\rangle, Q_{l-1}, l-1)$;
(3) for each $l \geq 1$, if $i_l \in \mathcal{D}_{Q_l}$ or $\Delta_{Q_{l-1}}(i_l) = 0$, then $\mathbf{r}_{Q_l} = \mathbf{r}_{Q_{l-1}}$ and $\mathcal{A}_{Q_{l+1}} = \mathcal{A}_{Q_l}$; else
  (a) (i) $r_{Q_l}(i_l) := r_{Q_{l-1}}(i_l) + \Delta_{Q_{l-1}}(i_l)$;
    (ii) for each session $i \in \mathcal{A}_{Q_l}$, $i \neq i_l$, such that $S_i \cap S_{i_l} \neq \emptyset$,

$$r_{Q_l}(i) := \min\{r_{Q_{l-1}}(i), r_{Q_{l-1}}(i_l) + \Delta_{Q_{l-1}}(i_l)\};$$

  (b) $\mathcal{A}_{Q_{l+1}} = \mathcal{A}_{Q_l} \setminus \mathsf{Term}(\langle G, \mathcal{S}\rangle, Q_l)$.

Call each state $Q_l$ in an execution $\alpha$, where $l \geq 1$, a *reachable state.* Say that $\mathsf{Alg}$ *computes the max-min fair rate vector* if for each execution of $\mathsf{Alg}$ a final state $Q$ is reachable such that $\mathbf{r}_Q$ is a max-min fair rate vector.

The number of `update` operations in any execution $\alpha$ of $\mathsf{Alg}$ is the number of state indices $l \geq 1$ such that $i_l \in \mathcal{A}_{Q_{l-1}}$. The *convergence complexity* of $\mathsf{Alg}$ on network $G$ with session set $\mathcal{S}$, denoted $\mathcal{U}_{\mathsf{Alg}}(\langle G, \mathcal{S}\rangle)$, is the number of operations in the execution of $\mathsf{Alg}$ on $G$ with $\mathcal{S}$. The *convergence complexity of* $\mathsf{Alg}$, denoted $\mathcal{U}_{\mathsf{Alg}}$, is the maximum, over all pairs $\langle G, \mathcal{S}\rangle$, of the convergence complexity of $\mathsf{Alg}$ on $G$ with $\mathcal{S}$.

## 3. Notation.

We collect here some notation that will be used in most of the following sections. For each edge $e$ and for any set of sessions $\mathcal{S}' \subseteq \mathcal{S}$, denote $\mathcal{S}' \mid e$ to be the set of sessions in $\mathcal{S}'$ traversing $e$. We will sometimes treat a session $S_i$ as the set of its links; so, for any edge $e$ traversed by $S_i$, we will write $e \in S_i$. For an edge $e$ and for a rate vector $\mathbf{r}$, denote by $\mathbf{r} \mid e$ the restriction of $\mathbf{r}$ to sessions traversing $e$. We will sometimes abuse notation by writing $\mathbf{r}_{\mathcal{A}_Q}$ and $\mathbf{r}_{\mathcal{D}_Q}$ to denote the restriction of $\mathbf{r}_Q$ to active and done sessions, respectively, in $Q$.

Define the *share-an-edge* relation on $\mathcal{S}$, denoted $\|_{\mathcal{S}}$, as follows. For any pair of sessions $S_i, S_j \in \mathcal{S}$, $S_i \|_{\mathcal{S}} S_j$ if $S_i$ and $S_j$ traverse a common edge. The transitive closure of $\|_{\mathcal{S}}$ is an equivalence relation on $\mathcal{S}$, which partitions $\mathcal{S}$ into equivalence classes $\mathcal{S}_1, \ldots, \mathcal{S}_c$, called *clusters*, where $1 \leq c \leq n$. The *session dependency* $d$ is the maximum size of a cluster. Call a cluster $\mathcal{S}_j$ an *active cluster* in state $Q$ if it contains at least one session that remains active in $Q$.

The set of *active edges* of the cluster $\mathcal{S}_j$ in state $Q$, denoted $AE_Q(\mathcal{S}_j)$, contains all edges of the network traversed by at least one active session in $Q$ that belongs to $\mathcal{S}_j$. The set of *active edges* of the network in state $Q$, denoted $AE_Q$, contains all edges of the network traversed by at least one active session in $Q$. The *residual capacity* of edge $e$ in state $Q$, denoted $resid_Q(e)$, is the difference between the capacity of $e$ and the total rate of sessions traversing $e$ in $Q$.

The set of *edges with minimum fair share* for cluster $\mathcal{S}_j$ in state $Q$ is defined as $\mathit{MFSE}_Q(\mathcal{S}_j) = \{e \in \mathit{AE}_Q(\mathcal{S}_j) \mid \mathit{FS}_Q(e) \leq \mathit{FS}_Q(e') \text{ for each } e' \in \mathit{AE}_Q(\mathcal{S}_j)\}$. The set of *edges with minimum fair share* in state $Q$, denoted $\mathit{MFSE}_Q$, is defined to be $\mathit{MFSE}_Q = \bigcup_{1 \leq j \leq c} \mathit{MFSE}_Q(\mathcal{S}_j)$.

An *execution fragment* $\alpha$ of Alg is a contiguous subsequence of some execution of Alg starting with the state $\mathit{first}(\alpha)$; if $\alpha$ is finite, it ends with the state $\mathit{last}(\alpha)$. For each execution (resp., execution fragment) $\alpha$ of Alg, the *schedule* $\sigma(\alpha)$ is the sequence of session indices in $\alpha$. If $\alpha$ is a finite execution fragment and $\alpha'$ is any execution fragment such that $\mathit{first}(\alpha') = \mathit{last}(\alpha)$, then $\alpha \cdot \alpha'$ is the concatenation of $\alpha$ and $\alpha'$, eliminating the duplicate occurrence of $\mathit{last}(\alpha) = \mathit{first}(\alpha')$.

For any index $l \geq 1$, the *preceding state* of $Q_l$ in execution $\alpha$, denoted $\overleftarrow{Q_l}$, is the state $Q_{l-1}$; for any index $l \geq 0$, the *successor state* of $Q_l$ in $\alpha$, denoted $\overrightarrow{Q_l}$, is the state $Q_{l+1}$. For any indices $l$ and $l' > l$, write $Q_l \xrightarrow{\alpha} Q_{l'}$ to denote that $Q_l$ precedes $Q_{l'}$; moreover, write $Q_l \overset{\alpha}{\longmapsto} Q_{l'}$ if additionally $Q_l$ and $Q_{l'}$ may coincide. For any index $l \geq 1$, we will sometimes abuse language and say that session $l$ is *scheduled in front* of state $Q_l$. For any state $Q$, denote $i_Q$ the session scheduled in front of $Q$. The *least schedule* for $Q_l$ in $\alpha$, denoted $\widehat{l}$, is the index of the earliest state following $Q_l$ by which all sessions that remain active in $Q_l$ have been scheduled at least once, or infinite if no such state exists. Define $\widehat{l} \mid e$ in the natural way.

## 4. Bottleneck algorithms.

In this section, we present basic properties of bottleneck algorithms, which will be useful in what follows. These properties refer to an execution $\alpha = Q_0, i_1, Q_1, \ldots, i_l, Q_l, \ldots$ of any bottleneck algorithm. To prove these properties, some more general properties are also proved in section 4.1.

### 4.1. Preliminaries.

We study how several quantities of interest change during an execution. We first state an immediate consequence of the definitions of allotted capacity and execution; we then prove a similar simple fact that will be helpful in later proofs.

LEMMA 4.1. *For each integer $l \geq 1$, and for any edge $e$,*

$$\mathit{allot}_{Q_l}(e) = \mathit{allot}_{Q_{l-1}}(e) + \sum_{i \in (\mathcal{A}_{Q_{l-1}} \setminus \mathcal{A}_{Q_l}) \mid e} r_{Q_{l-1}}(i).$$

LEMMA 4.2. *For any integers $l_0$ and $l$, $0 \leq l_0 < l$, and for any edge $e$,*

$$\sum_{i \in (\mathcal{A}_{Q_{l_0}} \setminus \mathcal{A}_{Q_l}) \mid e} r_{Q_{l_0}}(i) = \mathit{allot}_{Q_l}(e) - \mathit{allot}_{Q_{l_0}}(e).$$

*Proof.* Clearly, $i \in (\mathcal{A}_{Q_{l_0}} \setminus \mathcal{A}_{Q_l}) \mid e$ if and only if $i \in (\mathcal{D}_{Q_l} \setminus \mathcal{D}_{Q_{l_0}}) \mid e$. Hence, it follows that

$$\begin{aligned}
&\sum_{i \in (\mathcal{A}_{Q_{l_0}} \setminus \mathcal{A}_{Q_l}) \mid e} r_{Q_l}(i) \\
&= \sum_{i \in (\mathcal{D}_{Q_l} \setminus \mathcal{D}_{Q_{l_0}}) \mid e} r_{Q_l}(i) \\
&= \sum_{i \in \mathcal{D}_{Q_l} \mid e} r_{Q_l}(i) - \sum_{i \in \mathcal{D}_{Q_{l_0}} \mid e} r_{Q_l}(i) \qquad (\text{since } \mathcal{D}_{Q_{l_0}} \subseteq \mathcal{D}_{Q_l}) \\
&= \mathit{allot}_{Q_l}(e) - \mathit{allot}_{Q_{l_0}}(e),
\end{aligned}$$

as needed. ∎

We continue to prove that the saturation of an edge depends in a critical way on how rates of sessions traversing the edge compare to each other.

LEMMA 4.3. *For any integer $l_0 \geq 0$, assume that edge $e$ is active in state $Q_{l_0}$. Then, for each integer $l \geq l_0$, the following hold:*

(1) *if for each session $i \in \mathcal{A}_{Q_{l_0}} \mid e$, $r_{Q_l}(i) = FS_{Q_{l_0}}(e)$, then $e$ is saturated in $Q_l$;*
(2) *if for each session $i \in \mathcal{A}_{Q_{l_0}} \mid e$, $r_{Q_l}(i) < FS_{Q_{l_0}}(e)$, then $e$ is not saturated in $Q_l$;*
(3) *there exists no session $k \in \mathcal{A}_{Q_{l_0}} \mid e$ such that $r_{Q_l}(k) > FS_{Q_{l_0}}(e)$, while for each session $i \in \mathcal{A}_{Q_{l_0}} \mid e$, $i \neq k$, $r_{Q_l}(i) \geq FS_{Q_{l_0}}(e)$.*

*Proof.* We start by proving (1). Clearly,

$$\begin{aligned}
&\sum_{i\in\mathcal{A}_{Q_l}|e} r_{Q_l}(i)\\
&= \sum_{i\in\mathcal{A}_{Q_{l_0}}|e} r_{Q_l}(i) - \sum_{i\in(\mathcal{A}_{Q_{l_0}}\setminus\mathcal{A}_{Q_l})|e} r_{Q_l}(i) &&(\text{since } \mathcal{A}_{Q_l} \mid e \subseteq \mathcal{A}_{Q_{l_0}} \mid e)\\
&= \sum_{i\in\mathcal{A}_{Q_{l_0}}|e} FS_{Q_{l_0}}(e) - \big(allot_{Q_l}(e) - allot_{Q_{l_0}}(e)\big) &&(\text{by assumption and Lemma 4.2})\\
&= \left|\mathcal{A}_{Q_{l_0}} \mid e\right| \cdot FS_{Q_{l_0}}(e) - (allot_{Q_l}(e) - allot_{Q_{l_0}}(e))\\
&= c(e) - allot_{Q_{l_0}}(e) - (allot_{Q_l}(e) - allot_{Q_{l_0}}(e)) &&(\text{by definition of fair share})\\
&= c(e) - allot_{Q_l}(e),
\end{aligned}$$

as needed to establish that $e$ is saturated in state $Q_l$.

Condition (2) is proved in an almost identical way. (The only difference is that now the assumption for (2) and Lemma 4.2 imply that $\sum_{i\in\mathcal{A}_{Q_{l_0}}|e} r_{Q_l}(i) - \sum_{i\in(\mathcal{A}_{Q_{l_0}}\setminus\mathcal{A}_{Q_l})|e} r_{Q_l}(i) < \sum_{i\in\mathcal{A}_{Q_{l_0}}|e} FS_{Q_{l_0}}(e) - (allot_{Q_l}(e) - allot_{Q_{l_0}}(e).)$

We finally prove (3). Assume otherwise; so, there exists some session $k \in \mathcal{A}_{Q_{l_0}} \mid e$ such that $r_{Q_l}(k) > FS_{Q_{l_0}}(e)$, while for each $i \in \mathcal{A}_{Q_{l_0}} \mid e$, $i \neq k$, $r_{Q_l}(i) \geq FS_{Q_{l_0}}(e)$. Then

$$\begin{aligned}
&\sum_{i\in\mathcal{A}_{Q_l}|e} r_{Q_l}(i)\\
&\quad= \sum_{i\in\mathcal{A}_{Q_{l_0}}|e} r_{Q_l}(i) - \sum_{i\in(\mathcal{A}_{Q_{l_0}}\setminus\mathcal{A}_{Q_l})|e} r_{Q_l}(i) &&(\text{since } \mathcal{A}_{Q_l}|e \subseteq \mathcal{A}_{Q_{l_0}} \mid e)\\
&\quad= \sum_{i\in\mathcal{A}_{Q_{l_0}}|e} r_{Q_l}(i) - (allot_{Q_l}(e) - allot_{Q_{l_0}}(e)) &&(\text{by Lemma 4.2})\\
&\quad= r_{Q_l}(k) + \sum_{i\in\mathcal{A}_{Q_{l_0}}|e,i\neq k} r_{Q_l}(i) - allot_{Q_l}(e) + allot_{Q_{l_0}}(e)\\
&\quad> FS_{Q_{l_0}}(e) + \sum_{i\in\mathcal{A}_{Q_{l_0}}|e,i\neq k} FS_{Q_{l_0}}(e) - allot_{Q_l}(e) + allot_{Q_{l_0}}(e) &&(\text{by assumption})
\end{aligned}$$

$$
\begin{aligned}
&= \sum_{i \in \mathcal{A}_{Q_{l_0}} | e} FS_{Q_{l_0}}(e) - allot_{Q_l}(e) + allot_{Q_{l_0}}(e)) \\
&= \left|\mathcal{A}_{Q_{l_0}} \mid e\right| \cdot FS_{Q_{l_0}}(e) - allot_{Q_l}(e) + allot_{Q_{l_0}}(e) \\
&= c(e) - allot_{Q_{l_0}}(e) - allot_{Q_l}(e) + allot_{Q_{l_0}}(e) \qquad \text{(by definition of fair share)} \\
&= c(e) - allot_{Q_l}(e),
\end{aligned}
$$

so that $\sum_{i \in \mathcal{A}_{Q_l} | e} r_{Q_l}(i) > c(e) - allot_{Q_l}(e)$. This is a contradiction. ∎

The next claim follows directly from the definitions of bottleneck edge and fair share.

LEMMA 4.4. *Let $e$ and $e'$ be bottleneck edges for state $Q$ such that $(\mathcal{A}_Q \mid e) \bigcap (\mathcal{A}_Q \mid e') \neq \emptyset$. Then $FS_Q(e) = FS_Q(e')$.*

The following (easy to prove) claim is a direct consequence of the definitions of a bottleneck edge and a minimum fair share edge for some particular cluster.

LEMMA 4.5. *For any state $Q$ and cluster $\mathcal{S}_j$, consider an edge $e \in MFSE_Q(\mathcal{S}_j)$. Then $e$ is a bottleneck edge for $Q$.*

We are interested in algorithms for which a final state is reachable for each possible execution. Bottleneck algorithms (whether conservative or optimistic) enjoy a related, interesting property [15].

PROPOSITION 4.6 (Hayden [15]). *Assume that* Alg *is a bottleneck algorithm. Then, for any reachable final state $Q$ of* Alg*, $\mathbf{r}_Q$ is a max-min fair rate vector.*

Proposition 4.6 implies that in order to show that any given bottleneck algorithm computes the max-min fair rate vector, it suffices to prove that it reaches a final state. We continue with an interesting monotonicity property of fair share.

LEMMA 4.7 (Afek, Mansour, and Ostfeld [1]). *Assume that* Alg *is a bottleneck algorithm. Then, for each integer $l \geq 1$ and for any edge $e \in AE_{Q_l}$, $FS_{Q_l}(e) \geq FS_{Q_{l-1}}(e)$.*

**4.2. Properties of bottleneck edges.** We strengthen Lemma 4.7 for the special case of bottleneck edges. We present a collection of invariant properties for any edge that becomes bottleneck in the course of an execution of a bottleneck algorithm (whether conservative or optimistic).

PROPOSITION 4.8 (invariants of bottleneck edge). *Assume that* Alg *is bottleneck. For any integer $l_0 \geq 0$, fix any edge $e$ that is a bottleneck edge for $Q_{l_0}$. Then, for any integer $l \geq l_0$ such that $e \in AE_{Q_l}$, the following hold:*

(1) *$FS_{Q_l}(e) = FS_{Q_{l_0}}(e)$;*
(2) *$e$ is a bottleneck edge for $Q_l$;*
(3) *for any session $i \in (\mathcal{A}_{Q_{l_0}} \setminus \mathcal{A}_{Q_{l+1}}) \mid e$, $r_{Q_l}(i) = FS_{Q_{l_0}}(e)$.*

Roughly speaking, Proposition 4.8 establishes that no change in the fair share of an active edge occurs once the edge has become bottleneck, so that the edge remains bottleneck; moreover, the final rate of any active session traversing it is equal to this constant fair share.

*Proof.* The proof is by induction on $l$. For the basis case where $l = l_0$, condition (1) holds trivially, condition (2) holds by assumption, and condition (3) holds by definition of a bottleneck algorithm. Assume inductively that for some integer $l \geq l_0$, the claims hold for any integer $l'$ where $l_0 \leq l' < l$. For the induction step, we show that the claims hold for integer $l$. We start by proving condition (1). Clearly,

$$\begin{aligned}
&FS_{Q_l}(e)\\
&= \frac{c(e) - allot_{Q_l}(e)}{|\mathcal{A}_{Q_l} \mid e|}\\
&= \frac{c(e) - allot_{Q_{l-1}}(e) - \sum_{i\in(\mathcal{A}_{Q_{l-1}}\setminus\mathcal{A}_{Q_l})|e} r_{Q_{l-1}}(i)}{|\mathcal{A}_{Q_l} \mid e|} \qquad \text{(by Lemma 4.1)}\\
&= \frac{\left|\mathcal{A}_{Q_{l-1}} \mid e\right| FS_{Q_{l-1}}(e) - \sum_{i\in(\mathcal{A}_{Q_{l-1}}\setminus\mathcal{A}_{Q_l})|e} r_{Q_{l-1}}(i)}{|\mathcal{A}_{Q_l} \mid e|} \qquad \text{(by definition of fair share).}
\end{aligned}$$

Consider any session $i \in (\mathcal{A}_{Q_{l-1}} \setminus \mathcal{A}_{Q_l}) \mid e$. Since $\mathcal{A}_{Q_{l-1}} \subseteq \mathcal{A}_{Q_{l_0}}$, this implies that $i \in (\mathcal{A}_{Q_{l_0}} \setminus \mathcal{A}_{Q_l}) \mid e$, so that by the induction hypothesis (condition (3)), $r_{Q_{l-1}}(i) = FS_{Q_{l_0}}(e)$. So,

$$\begin{aligned}
&FS_{Q_l}(e)\\
&= \frac{\left|\mathcal{A}_{Q_{l-1}} \mid e\right| FS_{Q_{l-1}}(e) - \sum_{i\in(\mathcal{A}_{Q_{l-1}}\setminus\mathcal{A}_{Q_l})|e} FS_{Q_{l_0}}(e)}{|\mathcal{A}_{Q_l} \mid e|}\\
&= \frac{\left|\mathcal{A}_{Q_{l-1}} \mid e\right| FS_{Q_{l_0}}(e) - \sum_{i\in(\mathcal{A}_{Q_{l-1}}\setminus\mathcal{A}_{Q_l})|e} FS_{Q_{l_0}}(e)}{|\mathcal{A}_{Q_l} \mid e|} \qquad \text{(by the induction hypothesis)}\\
&= \frac{\left|\mathcal{A}_{Q_{l-1}} \mid e\right| FS_{Q_{l_0}}(e) - \left|(\mathcal{A}_{Q_{l-1}} \setminus \mathcal{A}_{Q_l}) \mid e\right| FS_{Q_{l_0}}(e)}{|\mathcal{A}_{Q_l} \mid e|}\\
&= \frac{\left(\left|\mathcal{A}_{Q_{l-1}} \mid e\right| - \left|(\mathcal{A}_{Q_{l-1}} \setminus \mathcal{A}_{Q_l}) \mid e\right|\right) FS_{Q_{l_0}}(e)}{|\mathcal{A}_{Q_l} \mid e|}\\
&= \frac{|\mathcal{A}_{Q_l} \mid e| \; FS_{Q_{l_0}}(e)}{|\mathcal{A}_{Q_l} \mid e|}\\
&= FS_{Q_{l_0}}(e),
\end{aligned}$$

which completes the proof of condition (1).

We now prove condition (2). Take any session $i \in \mathcal{A}_{Q_l} \mid e$. Clearly, $i \in \mathcal{A}_{Q_{l_0}} \mid e$. Since $e$ is a bottleneck edge for $Q_{l_0}$, $FS_{Q_{l_0}}(e) = MFS_{Q_{l_0}}(i) = \min_{e'\in S_i} FS_{Q_{l_0}}(e')$. Consider any edge $e' \in S_i$. Since $i \in \mathcal{A}_{Q_l} \mid e$, it follows that $e' \in AE_{Q_l}$; thus, by Lemma 4.7, $FS_{Q_l}(e') \geq FS_{Q_{l_0}}(e')$. Since $e'$ was chosen arbitrarily, this implies that $\min_{e'\in S_i} FS_{Q_l}(e') \geq \min_{e'\in S_i} FS_{Q_{l_0}}(e')$. It follows that $\min_{e'\in S_i} FS_{Q_l}(e') \geq FS_{Q_{l_0}}(e)$. By condition (1) above, this implies that $\min_{e'\in S_i} FS_{Q_l}(e') \geq FS_{Q_l}(e)$. Since $e \in S_i$, $\min_{e'\in S_i} FS_{Q_l}(e') \leq FS_{Q_l}(e)$. It follows that $FS_{Q_l}(e) = \min_{e'\in S_i} FS_{Q_l}(e')$. Since $i$ was chosen arbitrarily, this implies that $e$ is a bottleneck edge for $Q_l$, which completes the proof of condition (2).

We finally prove condition (3). Take any session $i \in (\mathcal{A}_{Q_{l_0}} \setminus \mathcal{A}_{Q_{l+1}}) \mid e$. Since $i \notin \mathcal{A}_{Q_{l+1}}$, there exists some integer $l'$, $l_0 < l' \leq l$, such that $i \in \mathsf{Term}(Q_{l'})$. Since $\mathsf{Term}$ is bottleneck, there exists some edge $e' \in S_i$ such that $e'$ is a bottleneck edge for $Q_{l'}$, and $r_{Q_{l'}}(i) = FS_{Q_{l'}}(e')$.

Either $l_0 < l' < l$ or $l' = l$. If $l_0 < l' < l$, then the induction hypothesis (condition (2)) implies that $e$ is a bottleneck edge for $Q_{l'}$; if, on the other hand, $l' = l$, then condition (2) above implies that $e$ is a bottleneck edge for $Q_{l'}$. Thus, in either case, $e$ is a bottleneck edge for $Q_{l'}$. Since both $e$ and $e'$ are bottleneck edges for state $Q'_l$ and $S_i \in (\mathcal{A}_{Q_{l'}} \mid e) \bigcap (\mathcal{A}_{Q_{l'}} \mid e')$, Lemma 4.4 implies that $FS_{Q_{l'}}(e) = FS_{Q_{l'}}(e')$.

Since $r_{Q_{l'}}(i) = FS_{Q_{l'}}(e')$, this implies that $r_{Q_{l'}}(i) = FS_{Q_{l'}}(e)$. Since $l \geq l'$ and $i \in \mathsf{Term}(Q_{l'})$, $r_{Q_l}(i) = r_{Q_{l'}}(i)$. Either $l_0 < l' < l$ or $l' = l$. If $l_0 < l' < l$, then the induction hypothesis (condition (1)) implies that $FS_{Q_{l'}}(e) = FS_{Q_{l_0}}(e)$; if, on the other hand, $l' = l$, then condition (1) implies that $FS_{Q_{l'}}(e) = FS_{Q_{l_0}}(e)$. Thus, in either case, $FS_{Q_{l'}}(e) = FS_{Q_{l_0}}(e)$. Hence, it follows that $r_{Q_l}(i) = FS_{Q_{l_0}}(e)$, which completes the proof of condition (3). □

**4.3. Properties of minimum fair share edges.** We start by proving a simple invariant property for any edge that becomes a minimum fair share edge for any particular cluster in the course of an execution of a bottleneck algorithm. We establish that the edge remains a minimum fair share edge (as long as it is active).

PROPOSITION 4.9 (invariant of minimum fair share edge). *Assume that* Alg *is bottleneck. For any integer $l_0 \geq 0$, fix any edge $e \in MFSE_{Q_{l_0}}(\mathcal{S}_j)$ for some active cluster $\mathcal{S}_j$ in $Q_{l_0}$. Then, for any integer $l \geq l_0$ such that $e \in AE_{Q_l}$, $e \in MFSE_{Q_l}(\mathcal{S}_j)$.*

*Proof.* Consider any edge $e' \in AE_{Q_l}(\mathcal{S}_j)$; clearly, $e' \in AE_{Q_{l_0}}(\mathcal{S}_j)$. Since $e \in MFSE_{Q_{l_0}}(\mathcal{S}_j)$, it follows that $FS_{Q_{l_0}}(e) \leq FS_{Q_{l_0}}(e')$. By Lemma 4.5, $e$ is a bottleneck edge for $Q_{l_0}$; thus, by Proposition 4.8 (condition (2)), $FS_{Q_l}(e) = FS_{Q_{l_0}}(e)$. By Lemma 4.7, $FS_{Q_{l_0}}(e') \leq FS_{Q_l}(e')$. So, $FS_{Q_l}(e) \leq FS_{Q_l}(e')$. Since $e'$ is arbitrary, it follows that $e \in MFSE_{Q_l}(\mathcal{S}_j)$. □

Similarly to Proposition 4.8, Proposition 4.9 holds for any bottleneck algorithm (whether conservative or optimistic) as well. However, the rest of the properties established in this section require the assumption of optimistic, bottleneck algorithms. We first prove a safety property for any edge that becomes a minimum fair share edge for any particular cluster during the execution of an optimistic, bottleneck algorithm.

PROPOSITION 4.10 (safety property of minimum fair share edge). *Assume that* Alg *is optimistic and bottleneck. For any integer $l_0 \geq 0$, fix any edge $e \in MFSE_{Q_{l_0}}(\mathcal{S}_j)$ for some active cluster $\mathcal{S}_j$ in $Q_{l_0}$. Consider any session $i \in \mathcal{A}_{Q_{l_0}} \mid e$ such that $r_{Q_{l_0}}(i) \geq FS_{Q_{l_0}}(e)$. Then, for any integer $l \geq l_0$, $r_{Q_l}(i) \geq FS_{Q_{l_0}}(e)$.*

Proposition 4.10 considers any (active) session traversing a minimum fair share edge; roughly speaking, it establishes that no decrease to its rate below this particular minimum fair share is possible if the rate is initially no less than the minimum fair share.

*Proof.* The proof is by induction on $l$. For the basis case where $l = l_0$, the claim holds by our assumption. Assume inductively that for some integer $l > l_0$, $r_{Q_{l-1}}(i) \geq FS_{Q_{l_0}}(e)$. For the induction step, we show that $r_{Q_l}(i) \geq FS_{Q_{l_0}}(e)$.

Assume first that $r_{Q_l}(i) \geq r_{Q_{l-1}}(i)$. By the induction hypothesis, this implies that $r_{Q_l}(i) \geq FS_{Q_{l_0}}(e)$, as needed. So assume that $r_{Q_l}(i) < r_{Q_{l-1}}(i)$. By definition of execution and `update` operation, this implies that $S_i$ intersects the session $S_{i_l}$, scheduled in front of state $Q_l$; moreover, $r_{Q_l}(i) = r_{Q_l}(i_l)$. Let $e'$ be an edge such that $\Delta_{Q_{l-1}}(i_l) = \Delta_{Q_{l-1}}(i_l, e')$. We prove the following.

LEMMA 4.11. $r_{Q_l}(i_l) \geq FS_{Q_l}(e')$.

*Proof.* Assume otherwise; so, $r_{Q_l}(i_l) < FS_{Q_l}(e')$. By definition of the `update` operation, $e'$ is saturated in $Q_l$; moreover, for any session $k \in \mathcal{A}_{Q_{l_0}} \mid e'$, $r_{Q_l}(i_l) \geq r_{Q_l}(k)$. Thus, $FS_{Q_l}(e') > r_{Q_l}(k)$. Lemma 4.3 (condition (2)) implies that $e'$ is not saturated in $Q_l$. This is a contradiction. □

Since $r_{Q_l}(i) = r_{Q_l}(i_l)$, Lemma 4.11 implies that $r_{Q_l}(i) \geq FS_{Q_l}(e')$. Also, by Lemma 4.7, $FS_{Q_l}(e') \geq FS_{Q_{l_0}}(e')$, so that $r_{Q_l}(i) \geq FS_{Q_{l_0}}(e')$. Since $e \in MFSE_{Q_{l_0}}(\mathcal{S}_j)$, $FS_{Q_{l_0}}(e') \geq FS_{Q_{l_0}}(e)$. It follows that $r_{Q_l}(i) \geq FS_{Q_{l_0}}(e)$, as needed. □

The next claim complements Proposition 4.10 by giving a corresponding liveness property.

Proposition 4.12 (liveness property of minimum fair share edge). *Assume that* Alg *is optimistic and bottleneck. For any integer $l_0 \geq 0$, fix any edge $e \in \mathit{MFSE}_{Q_{l_0}}(\mathcal{S}_j)$ for some active cluster $\mathcal{S}_j$ in $Q_{l_0}$, such that $\widehat{l_0} \mid e < \infty$. Consider any session $i \in \mathcal{A}_{Q_{l_0}} \mid e$. Then, for any integer $l \geq \widehat{l_0} \mid e$, $r_{Q_l}(i) \geq FS_{Q_{l_0}}(e)$.*

Proposition 4.12 considers any active session traversing a minimum fair share edge; roughly speaking, it establishes that eventually, once all active sessions traversing this minimum fair share edge have been scheduled at least once, the rate of the session will be no less than this particular minimum fair share.

*Proof.* We start with an informal outline of the proof. We consider the point of the execution following state $Q_{l_0}$ where session $i$ is scheduled; clearly, that point comes no later than when all sessions have been scheduled at least once. We establish that at this point, the rate of $S_i$ is no less than the fair share of edge $e$ in state $Q_{l_0}$. We also argue that $e$ remains a minimum fair share edge beyond state $Q_{l_0}$; this allows us to exploit the safety property of minimum fair share edges in order to argue that the rate of $S_i$ will subsequently remain no less than the fair share of $e$ in state $Q_{l_0}$. We now present the details of the formal proof.

Since $e \in \mathit{MFSE}_{Q_{l_0}}(\mathcal{S}_j)$, Lemma 4.5 implies that $e$ is a bottleneck edge for state $Q_{l_0}$. Since $i \in \mathcal{A}_{Q_{l_0}} \mid e$, it follows by definition of $\widehat{l_0} \mid e$ that there exists a least index $l'$, $l_0 < l' \leq \widehat{l_0} \mid e$, such that $i$ is scheduled in front of state $Q_{l'}$. We proceed by case analysis.

1. Assume first that $i$ is not active in state $Q_{l'}$. Since $i \in \mathcal{A}_{Q_{l_0}}$, there exists an index $l''$, $l_0 \leq l'' < l'$, such that $i \in \mathsf{Term}(Q_{l''})$. Since $\mathsf{Term}$ is bottleneck, it follows that there exists some edge $e'$ traversed by session $i$ that is a bottleneck edge for state $Q_{l''}$, and $r_{Q_{l''}}(i) = FS_{Q_{l''}}(e')$. Since $i \in \mathcal{A}_{Q_{l''}}$ and $i$ traverses $e$, $e$ is an active edge at $Q_{l''}$. By Proposition 4.8 (conditions (1) and (2)), $FS_{Q_{l''}}(e) = FS_{Q_{l_0}}(e)$, and $e$ is a bottleneck edge for $Q_{l''}$. By Lemma 4.4, $FS_{Q_{l''}}(e') = FS_{Q_{l''}}(e)$, so that $FS_{Q_{l''}}(e') = FS_{Q_{l_0}}(e)$. It follows that $r_{Q_{l''}}(i) = FS_{Q_{l_0}}(e)$. Now take any integer $l \geq \widehat{l_0} \mid e$. Clearly, $l \geq l''$. Since $i \in \mathsf{Term}(Q_{l''})$, $r_{Q_l}(i) = r_{Q_{l''}}(i) = FS_{Q_{l_0}}(e)$, which establishes the claim in this case.
2. Assume now that $i$ is active in state $Q_{l'}$. Since $i$ traverses edge $e$, it follows that $e$ is active in state $Q_{l'}$. By Proposition 4.8 (conditions (1) and (2)), $FS_{Q_{l'}}(e) = FS_{Q_{l_0}}(e)$, and $e$ is a bottleneck edge for $Q_{l'}$. We prove the following.
   Lemma 4.13. $r_{Q_{l'}}(i) \geq FS_{Q_{l_0}}(e)$.
   *Proof.* Assume, by way of contradiction, that $r_{Q_{l'}}(i) < FS_{Q_{l_0}}(e)$. Let $e'$ be an edge such that $\Delta_{Q_{l'}}(i) = \Delta_{Q_{l'}}(i, e')$. By definition of the `update` operation, $e'$ is saturated in state $Q_{l'}$. Since $e$ is a bottleneck edge for state $Q_{l'}$, and $i$ traverses both $e$ and $e'$, $FS_{Q_{l'}}(e) \leq FS_{Q_{l'}}(e')$. Since $FS_{Q_{l'}}(e) = FS_{Q_{l_0}}(e)$, this implies that $FS_{Q_{l_0}}(e) \leq FS_{Q_{l'}}(e')$. Since $r_{Q_{l'}}(i) < FS_{Q_{l_0}}(e)$, it follows that $r_{Q_{l'}}(i) < FS_{Q_{l'}}(e')$. By definition of the `update` operation, for any session $k \in \mathcal{A}_{Q_{l'}} \mid e$, $r_{Q_{l'}}(k) \leq r_{Q_{l'}}(i)$, so that $r_{Q_{l'}}(k) < FS_{Q_{l'}}(e')$. It follows by Lemma 4.3 (condition (2)) that $e'$ is not saturated in state $Q_{l'}$. This is a contradiction. □
   Now take any integer $l \geq \widehat{l_0} \mid e$. Clearly, $l \geq l'$. Since $e$ is a minimum fair share edge for $Q_{l_0}$, Proposition 4.9 implies that $e$ is a minimum fair share edge for $Q_{l'}$ as well. Moreover, by Lemma 4.13, $r_{Q_{l'}}(i) \geq FS_{Q_{l_0}}(e)$. Since $FS_{Q_{l'}}(e) = FS_{Q_{l_0}}(e)$, this implies that $r_{Q_{l'}}(i) \geq FS_{Q_{l'}}(e)$. It follows by Proposition 4.10 (taking $l'$ for $l_0$) that $r_{Q_l}(i) \geq FS_{Q_{l_0}}(e)$, which establishes the claim in this case.

The proof of Proposition 4.12 is now complete. ∎

**4.4. Termination properties.** The first property considers active sessions in any particular cluster that traverse a minimum fair share edge; it is established that once each such session has been scheduled at least once, all of these sessions must have become done.

PROPOSITION 4.14 (termination of all sessions). *Assume that* Alg *is optimistic and bottleneck. For any integer $l_0 \geq 0$, fix any edge $e \in MFSE_{Q_{l_0}}(\mathcal{S}_j)$ for some active cluster $\mathcal{S}_j$ in $Q_{l_0}$ such that $\widehat{l_0} \mid e < \infty$. Then, for any session $i \in \mathcal{A}_{Q_{l_0}} \mid e$, $i \in \mathcal{D}_{\overrightarrow{Q_{\widehat{l_0}|e}}}$.*

*Proof.* We start with an informal outline of the proof. We consider any session active in state $Q_{l_0}$, and we argue that after all sessions have been scheduled at least once, the session will receive a rate equal to the fair share of $e$ in $Q_{l_0}$. We will exploit the fact that $e$ is a bottleneck edge for $Q_{l_0}$ in order to argue that $e$ remains bottleneck subsequently, and that its fair share does not change. Since Alg is a bottleneck algorithm, this will be sufficient for deducing that the session has reached its final rate. We now present the details of the formal proof.

Fix any session $i \in \mathcal{A}_{Q_{l_0}} \mid e$. We start by proving the following.

LEMMA 4.15. *$r_{Q_{\widehat{l_0}|e}}(i) = FS_{Q_{l_0}}(e)$.*

*Proof.* Assume, by way of contradiction, that $r_{Q_{\widehat{l_0}|e}}(i) \neq FS_{Q_{l_0}}(e)$. By Proposition 4.12, $r_{Q_{\widehat{l_0}|e}}(i) \geq FS_{Q_{l_0}}(e)$. It follows that $r_{Q_{\widehat{l_0}|e}}(i) > FS_{Q_{l_0}}(e)$. We proceed by case analysis.

Assume first that there exists no session $k \in \mathcal{A}_{Q_{l_0}} \mid e$ with $k \neq i$; thus, $\mathcal{A}_{Q_{l_0}} \mid e = \{i\}$. Denote by $Q_l$ the latest state in execution $\alpha$, such that $Q_{l_0} \xmapsto{\alpha} Q_l \xmapsto{\alpha} Q_{\widehat{l_0}|e}$, and $\mathcal{A}_{Q_l} \neq \emptyset$. Thus, $\mathcal{A}_{Q_l} \mid e = \mathcal{A}_{Q_{l_0}} \mid e$ and $allot_{Q_l}(e) = allot_{Q_{l_0}}(e)$. Clearly,

$$\begin{aligned}
&\sum_{k \in \mathcal{A}_{Q_l}|e} r_{Q_l}(k) \\
&\quad = r_{Q_l}(i) && (\text{since } \mathcal{A}_{Q_{l_0}} \mid e = \{i\}) \\
&\quad = r_{Q_{\widehat{l_0}|e}}(i) && (\text{by definition of state } Q_l) \\
&\quad > FS_{Q_{l_0}}(e) && (\text{by assumption}) \\
&\quad = c(e) - allot_{Q_{l_0}}(e) && (\text{by definition of fair share and since } \left|\mathcal{A}_{Q_{l_0}} \mid e\right| = 1) \\
&\quad = c(e) - allot_{Q_l}(e).
\end{aligned}$$

This is a contradiction.

Assume now that there exists some session $k \in \mathcal{A}_{Q_{l_0}} \mid e$ with $k \neq i$. By Proposition 4.12, $r_{Q_{\widehat{l_0}|e}}(k) \geq FS_{Q_{l_0}}(e)$. Together with $r_{Q_{\widehat{l_0}|e}}(i) > FS_{Q_{l_0}}(e)$, this implies a contradiction to Lemma 4.3 (condition (3)), and the proof is now complete. ∎

We continue with the proof of Proposition 4.14. In case $i \in \mathcal{D}_{Q_l}$ for some state $Q_l$ in $\alpha$ such that $Q_{l_0} \xrightarrow{\alpha} Q_l \xmapsto{\alpha} Q_{\widehat{l_0}|e}$, the definition of execution implies that $i \in \mathcal{D}_{\overrightarrow{Q_{\widehat{l_0}|e}}}$. So, assume that for each state $Q_l$ in execution $\alpha$ such that $Q_{l_0} \xrightarrow{\alpha} Q_l \xmapsto{\alpha} Q_{\widehat{l_0}|e}$, $i \in \mathcal{A}_{Q_l}$. Denote by $Q_l$ the latest state in execution $\alpha$ such that both $Q_{l_0} \xrightarrow{\alpha} Q_l \xmapsto{\alpha} Q_{\widehat{l_0}|e}$ and $r_{Q_l}(i) = r_{Q_{\widehat{l_0}|e}}(i)$.

Since $e \in MFSE_{Q_{l_0}}$, Lemma 4.5 implies that $e$ is a bottleneck edge for $Q_{l_0}$. Since $i \in \mathcal{A}_{Q_l}$ and $i$ traverses edge $e$, it follows that $\mathcal{A}_{Q_l} \mid e \neq \emptyset$. Hence, Proposition 4.8 (conditions (1) and (2)) implies that $FS_{Q_l}(e) = FS_{Q_{l_0}}(e)$, and $e$ is a bottleneck edge for $Q_l$. By Lemma 4.15, $r_{Q_{\widehat{l_0}|e}}(i) = FS_{Q_{l_0}}(e)$. It follows that $r_{Q_l}(i) = FS_{Q_l}(e)$.

In total, $e$ is a bottleneck edge for state $Q_l$, traversed by session $i$ for which $r_{Q_l}(i) = FS_{Q_l}(e)$. Since $\mathsf{Alg}$ is bottleneck, it follows that $i \in \mathcal{D}_{\overrightarrow{Q_l}}$. Since $Q_l \to \overrightarrow{Q_{\widehat{l_0}|e}}$, it follows that $i \in \mathcal{D}_{\overrightarrow{Q_{\widehat{l_0}|e}}}$. □

The final termination property is a direct consequence of Proposition 4.14. In essence, we establish that scheduling any sequence of sessions that includes all currently active ones must result in finalizing the rate of *at least one* active session per cluster.

PROPOSITION 4.16 (termination of at least one session per cluster). *Assume that* $\mathsf{Alg}$ *is optimistic and bottleneck. For any integer* $l_0 \geq 0$ *such that* $\mathcal{A}_{Q_{l_0}} \neq \emptyset$ *and* $\widehat{l_0} < \infty$*, fix any active cluster* $\mathcal{S}_j$ *in* $Q_{l_0}$*. Then there exists some session* $S_i \in \mathcal{S}_j \cap \mathcal{A}_{Q_{l_0}}$ *such that* $S_i \in \mathcal{D}_{\overrightarrow{Q_{\widehat{l_0}}}}$*.*

*Proof.* Since $\mathcal{S}_j$ is active in $Q_{\mathcal{A}_{Q_{l_0}}}$, it follows that $MFSE_{Q_{l_0}}(\mathcal{S}_j) \neq \emptyset$. Fix any edge $e \in MFSE_{Q_{l_0}}(\mathcal{S}_j)$, and consider any session $i \in \mathcal{A}_{Q_{l_0}} \mid e$. By Proposition 4.14, $i \in \mathcal{D}_{\overrightarrow{Q_{\widehat{l_0}}}}$. □

## 5. Upper bounds.

**5.1. Oblivious algorithms.** This section presents the algorithm $\mathsf{RoundRobin}$ and shows the following.

THEOREM 5.1 (upper bound for oblivious algorithms). $\mathsf{RoundRobin}$ *computes the max-min fair rate vector within* $\frac{dn}{2} + \frac{n}{2}$ `update` *operations.*

The scheduler of $\mathsf{RoundRobin}$ conducts scheduling rounds. In each round, each of the $n$ sessions is scheduled in round-robin order. Moreover, $\mathsf{RoundRobin}$ is bottleneck. By definition of $\mathsf{RoundRobin}$, each session is scheduled once in each round. Thus, Proposition 4.16 implies that at least one session per cluster becomes done in each round. Since each cluster contains at most $d$ sessions, all sessions are done after $d$ rounds, whence the network enters a final state. So, Proposition 4.6 immediately implies that $\mathsf{RoundRobin}$ computes the max-min fair rate vector.

We now establish an upper bound on the convergence complexity of $\mathsf{RoundRobin}$. Since at least one session per cluster becomes done in each round, at most $|\mathcal{S}_j| - l + 1$ `update` operations are executed in round $l$, $1 \leq l \leq |\mathcal{S}_j|$, on sessions in any cluster $\mathcal{S}_j$. Summing up over all clusters and rounds yields that

$$\begin{aligned}
\mathcal{U}_{\mathsf{RoundRobin}} &\leq \sum_{j\geq 1} \sum_{1\leq l\leq d} \max\{0, (|\mathcal{S}_j| - l + 1)\} \\
&\leq \sum_{j\geq 1} \sum_{1\leq l\leq |\mathcal{S}_j|} (|\mathcal{S}_j| - l + 1) \\
&= \sum_{j\geq 1} \frac{|\mathcal{S}_j|(\mathcal{S}_j + 1)}{2} \\
&\leq \frac{dn}{2} + \frac{n}{2}.
\end{aligned}$$

$\mathsf{RoundRobin}$ extends naturally to a class of bottleneck algorithms $d$-$\mathsf{Epoch}$. The scheduler $d$-$\mathsf{EpochSched} = d$-$\mathsf{EpochSched}_1 \ldots d$-$\mathsf{EpochSched}_d$ $d$-$\mathsf{EpochSched}'$ of algorithm $d$-$\mathsf{Epoch}$ is a sequence such that for each index $r$, $1 \leq r \leq d$, all sessions are included in $d$-$\mathsf{EpochSched}_r$. An argument identical to the one applied to $\mathsf{RoundRobin}$ immediately implies the following.

THEOREM 5.2 (upper bound for oblivious algorithms). $d$-$\mathsf{Epoch}$ *computes the max-min fair rate vector within* $\frac{dn}{2} + \frac{n}{2}$ `update` *operations.*

**5.2. Nonoblivious algorithms.** This section presents the algorithm $\mathsf{Linear}$ and shows the following.

THEOREM 5.3 (upper bound for nonoblivious algorithms). $\mathsf{Linear}$ *computes the max-min fair rate vector within exactly* $n$ `update` *operations.*

The scheduler of $\mathsf{Linear}$ maintains an active edge of minimum fair share and schedules all active sessions traversing it in any order. Once it finishes, it chooses any other (active) edge of minimum fair share, and so on. Moreover, $\mathsf{Linear}$ is bottleneck.

Consider any state $Q_{l_0}$ and an arbitrary edge $e \in \mathit{MFSE}_{Q_{l_0}}$. Clearly, $e \in \mathit{MFSE}_{Q_{l_0}}(\mathcal{S}_j)$ for some cluster $\mathcal{S}_j$. By definition of $\mathsf{Linear}$, each session traversing $e$ is scheduled exactly once, so that the state $Q_{\widehat{l_0|e}}$ is reached; by Proposition 4.14, each such session is done in state $Q_{\widehat{l_0|e}}$. It follows that all sessions eventually become done and a final state is reached. Hence, Proposition 4.6 immediately implies that $\mathsf{Linear}$ computes the max-min fair rate vector.

$\mathsf{Linear}$ incurs $n$ `update` operations. Recall that all rates are initially zero. Since all capacities exceed zero, all rates in a max-min fair rate vector exceed zero as well. Since each `update` increases the rate of exactly one session, it follows that at least $n$ `update` operations are needed, so that $\mathcal{U}_{\mathsf{Alg}} \geq n$ for every $\mathsf{Alg}$. Thus, $\mathsf{Linear}$ is optimal.

**6. Network construction.** We present a generic, combinatorial construction of a network associated with any sequence $\mathsf{Seq} = i_1, i_2, \ldots$, where for each $l \geq 1$, $i_l \in [n]$. For any sequence $\mathsf{Seq}$, denote $|\mathsf{Seq}|$ to be the *length* of $\mathsf{Seq}$; an infinite sequence has infinite length. For a sequence $\mathsf{Seq}$ of session indices, and for any set of sessions $\mathcal{S}' \subseteq \mathcal{S}$, denote by $\mathsf{Seq} \mid \mathcal{S}'$ the *restriction* of $\mathsf{Seq}$ to indices of sessions in $\mathcal{S}'$. Denote by $\mathsf{Seq} \uparrow \mathcal{S}'$ the shortest prefix of $\mathsf{Seq} \mid \mathcal{S}'$ that includes the indices of all sessions in $\mathcal{S}'$, in the order they appear in this prefix of $\mathsf{Seq} \mid \mathcal{S}'$ and with no repetitions; in case no such prefix exists, $\mathsf{Seq} \uparrow \mathcal{S}'$ results from $\mathsf{Seq} \mid \mathcal{S}'$ by removing repetitions, while, however, preserving the order of the indices. Denote by $\mathsf{Seq} \downarrow \mathcal{S}'$ the remaining suffix of $\mathsf{Seq} \mid \mathcal{S}'$. For example, if $\mathsf{Seq} = 1, 5, 4, 2, 1, 3, 3, 3, 5, 4$ and $\mathcal{S}' = \{1, 3, 4\} \subset \{1, 2, 3, 4, 5\}$, then $\mathsf{Seq} \mid \mathcal{S}' = 1, 4, 1, 3, 3, 3, 4$, $\mathsf{Seq} \uparrow \mathcal{S}' = 1, 4, 3$, and $\mathsf{Seq} \downarrow \mathcal{S}' = 3, 3, 4$.

Fix any even integer $d$, and choose any integer $n$ that is a multiple of $d$.[3] We construct a network $G = G(\mathsf{Seq}) = (V(\mathsf{Seq}), E(\mathsf{Seq}))$, as a function of $\mathsf{Seq}$, with a set of sessions $\mathcal{S} = \{S_1, S_2, \ldots, S_n\}$ laid out on $G$. For assigning capacities to network edges, we will use two (finite) sequences of real numbers, $b$ and $p$ (for *bottom* and *potential*, resp.), each of length $\frac{d}{2}$, defined recursively as follows.

- $b_0 = p_0 = 0$, $b_1 = 0$, and $p_1 = 2^p$ for some integer $p \geq d$;
- for each index $r$, $1 < r \leq \frac{d}{2}$, $b_r = b_{r-1} + \frac{p_{r-1}}{2}$ and $p_r = \frac{p_{r-1}}{4}$.

Partition $\mathcal{S}$ into $\frac{n}{d}$ clusters $\mathcal{S}_1, \mathcal{S}_2, \ldots, \mathcal{S}_{n/d}$ so that for each $j$, $1 \leq j \leq \frac{n}{d}$, $\mathcal{S}_j$ contains sessions $S_{(j-1)d+1}, S_{(j-1)d+2}, \ldots, S_{(j-1)d+d}$; notice that $|\mathcal{S}_j| = d$. For each cluster $\mathcal{S}_j$, we construct a network $G_j = G_j(\mathsf{Seq} \mid \mathcal{S}_j) = (V_j(\mathsf{Seq} \mid \mathcal{S}_j), E_j(\mathsf{Seq} \mid \mathcal{S}_j))$, with $\mathcal{S}_j$ laid out on $G_j(\mathsf{Seq} \mid \mathcal{S}_j)$, so that

$$G(\mathsf{Seq}) = \left( \bigcup_{1 \leq j \leq n/d} V_j(\mathsf{Seq} \mid \mathcal{S}_j), \bigcup_{1 \leq j \leq n/d} E_j(\mathsf{Seq} \mid \mathcal{S}_j) \right);$$

thus, each individual network $G_j$ is a function of the sequence $\mathsf{Seq} \mid \mathcal{S}_j$, and the network $G$ is the resulting composition.

[3]Standard "padding" techniques can be used to handle the case where $n$ is not a multiple of $d$.

The construction of the network $G_j$ proceeds in a sequence of $\frac{d}{2}$ *epochs*; in epoch $r$, $1 \leq r \leq \frac{d}{2}$, the network $G_j^{(r)} = (V_j^{(r)}, E_j^{(r)})$ is constructed, so that $G_j = (\bigcup_{1\leq r\leq d/2} V_j^{(r)}, \bigcup_{1\leq r\leq d/2} E_j^{(r)})$; thus, the network $G_j$ is the composition of the individual networks $G_j^{(r)}$.

For each $r$, $1 \leq r \leq \frac{d}{2}$, the construction of $G_j^{(r)}$ uses $b_r$ and $p_r$ as parameters. It also uses the following sets and sequences:

- a set of indices $\mathcal{I}_j^{(r)} \subseteq \mathcal{S}_j$ such that $|\mathcal{I}_j^{(r)}| = d - 2(r-1)$;
- a sequence $\mathsf{Seq}_j^{(r)}$, which is a suffix of $\mathsf{Seq} \mid \mathcal{S}_j$;
- a set $\{i_f^{(r)}, i_l^{(r)}\} \subseteq \mathcal{I}_j^{(r)}$; roughly speaking, $i_f^{(r)}$ and $i_l^{(r)}$ will be defined to be the first and last indices, respectively, of $\mathsf{Seq}_j^{(r)} \uparrow \mathcal{I}_j^{(r)}$, or some of them will be set to arbitrary indices from $\mathcal{I}_j^{(r)}$ in case $|\mathsf{Seq}_j^{(r)} \uparrow \mathcal{I}_j^{(r)}| < 2$.

These sets and sequences are inductively defined as follows. For the basis case where $r = 1$, $\mathcal{I}_j^{(1)} := \mathcal{S}_j$, $\mathsf{Seq}_j^{(1)} := \mathsf{Seq} \mid \mathcal{S}_j$, and the set $\{i_f^{(1)}, i_l^{(1)}\}$ is defined as follows:

(1) First, assume that $\mathsf{Seq}_j^{(1)} = \lambda$, the *empty* sequence, so that $\mathsf{Seq}_j^{(1)} \uparrow \mathcal{I}_j^{(1)} = \lambda$; then fix $i_f^{(1)}$ and $i_l^{(1)}$ to be any arbitrary indices in $\mathcal{I}_j^{(1)}$.

(2) Now assume that $\mathsf{Seq}_j^{(1)} \neq \lambda$, so that $\mathsf{Seq}_j^{(1)} \uparrow \mathcal{I}_j^{(1)} \neq \lambda$; there are two cases to consider.

  (a) First, take $|\mathsf{Seq}_j^{(1)} \uparrow \mathcal{I}_j^{(1)}| = 1$; then $i_f^{(1)} := \mathsf{Seq}_j^{(1)} \uparrow \mathcal{I}_j^{(1)}$, and fix $i_l^{(1)}$ to be any arbitrary index in $\mathcal{I}_j^{(1)} \setminus \{i_f^{(1)}\}$.

  (b) Finally, take $|\mathsf{Seq}_j^{(1)} \uparrow \mathcal{I}_j^{(1)}| > 1$, so that $\mathsf{Seq}_j^{(1)} \uparrow \mathcal{I}_j^{(1)} = i_f, \ldots, i_l$; then $i_f^{(1)} := i_f$ and $i_l^{(1)} := i_l$.

Informally, $\mathcal{I}_j^{(1)}$ contains indices of all sessions in cluster $\mathcal{S}_j$, while $\mathsf{Seq}_j^{(1)}$ is the restriction of $\mathsf{Seq}$ to indices of sessions in cluster $\mathcal{S}_j$; moreover, $i_f^{(1)}$ and $i_l^{(1)}$ are the indices of sessions in cluster $\mathcal{S}_j$ appearing first and last, respectively, in $\mathsf{Seq}_j^{(1)} \uparrow \mathcal{I}_j^{(1)}$ or some of them will be set to be arbitrary indices if $\mathsf{Seq}_j^{(1)} \uparrow \mathcal{I}_j^{(1)}$ misses any such indices.

Assume inductively that we have defined $\mathcal{I}_j^{(r-1)}$, $\mathsf{Seq}_j^{(r-1)}$, and $\{i_f^{(r-1)}, i_l^{(r-1)}\}$ for some integer $r$, where $2 \leq r \leq \frac{d}{2}$. For the induction step, we show how to construct $\mathcal{I}_j^{(r)}$, $\mathsf{Seq}_j^{(r)}$, and $\{i_f^{(r)}, i_l^{(r)}\}$. Define $\mathcal{I}_j^{(r)} := \mathcal{I}_j^{(r-1)} \setminus \{i_f^{(r-1)}, i_l^{(r-1)}\}$, $\mathsf{Seq}_j^{(r)} := (\mathsf{Seq}_j^{(r-1)} \downarrow \mathcal{I}_j^{(r-1)}) \mid \mathcal{I}_j^{(r)}$, and the set $\{i_f^{(r)}, i_l^{(r)}\}$ is defined through a case analysis identical to the one for the basis case.

(1) Assume first that $\mathsf{Seq}_j^{(r)} = \lambda$, so that $\mathsf{Seq}_j^{(r)} \uparrow \mathcal{I}_j^{(r)} = \lambda$; then fix $i_f^{(r)}, i_l^{(r)}$ to be any arbitrary indices in $\mathcal{I}_j^{(r)}$.

(2) Now assume that $\mathsf{Seq}_j^{(r)} \neq \lambda$, so that $\mathsf{Seq}_j^{(r)} \uparrow \mathcal{I}_j^{(r)} \neq \lambda$; there are two cases to consider.

  (a) First, take $|\mathsf{Seq}_j^{(r)} \uparrow \mathcal{I}_j^{(r)}| = 1$; then $i_f^{(r)} := \mathsf{Seq}_j^{(r)} \uparrow \mathcal{I}_j^{(r)}$, and fix $i_l^{(r)}$ to be any arbitrary index in $\mathcal{I}_j^{(r)}$;

  (b) Finally, take $|\mathsf{Seq}_j^{(r)} \uparrow \mathcal{I}_j^{(r)}| > 1$, so that $\mathsf{Seq}_j^{(r)} \uparrow \mathcal{I}_j^{(r)} = i_f, \ldots, i_l$; then $i_f^{(r)} := i_f$ and $i_l^{(r)} := i_l$.

Informally, $\mathcal{I}_j^{(r)}$ is obtained by removing $i_f^{(r-1)}$ and $i_l^{(r-1)}$ from $\mathcal{I}_j^{(r-1)}$, while $\mathsf{Seq}_j^{(r)}$ results from $\mathsf{Seq}_j^{(r-1)}$ by chopping off its shortest prefix that includes all indices in

$\mathcal{I}_j^{(r-1)}$ and restricting the remaining suffix to indices in $\mathcal{I}_j^{(r)}$; moreover, $i_f^{(r)}$ and $i_l^{(r)}$ are the indices of sessions in $\mathcal{I}_j^{(r)}$ that appear first and last, respectively, in this suffix, or some of them will be set to arbitrary indices in case this suffix misses any such indices.

Since two different sessions are extracted from $\mathcal{S}_j$ in each of the $\frac{d}{2}$ epochs, all $d$ sessions in $\mathcal{S}_j$ will eventually be extracted. We now describe the construction of $G_j^{(r)}$, $1 \leq r \leq \frac{d}{2}$:

- sessions $i_f^{(r)}$ and $i_l^{(r)}$ traverse some edge $e_{i_l}^{(r)}$ with $c(e_{i_l}^{(r)}) = 2b_r + \frac{p_r}{2}$;
- for each $i \in \mathcal{I}_j^{(r)} \setminus \{i_f^{(r)}, i_l^{(r)}\}$, sessions $i_f^{(r)}$ and $i$ traverse some edge $e_i^{(r)}$ with $c(e_i^{(r)}) = 2b_r + p_r$.

Informally, $i_f^{(r)}$ shares an edge with every other session in $\mathcal{I}_j^{(r)}$; the capacity of the edge shared with $i_l^{(r)}$ is the smallest, while all other capacities are equal. All other sessions traverse only the edge shared with $i_f^{(r)}$. We finally state an easy to prove property of the construction.

LEMMA 6.1. *For each integer $r$, where $1 < r \leq \frac{d}{2}$, for each index $r'$ where $1 \leq r' < r$, let $e^{(r)}$ and $e^{(r')}$ be any edges in $E_j^{(r)}$ and $E_j^{(r')}$, respectively. Then $c(e^{(r)}) > c(e^{(r')})$.*

*Example.* Fix $d = 6$ and choose $n = 12$. Consider the (infinite) elevator sequence

$$\mathsf{ElevSched} = 1, 2, \ldots, 11, 12, 12, 11, \ldots, 2, 1, \ldots, 1, 2, \ldots, 11, 12, 12, 11, \ldots, 2, 1, \ldots .$$

We construct the network $G = G(\mathsf{ElevSched}) = (V(\mathsf{ElevSched}), E(\mathsf{ElevSched}))$ as a function of $\mathsf{ElevSched}$, with a set of sessions $\mathcal{S} = \{S_1, \ldots, S_{12}\}$ laid out on $G$. Partition $\mathcal{S}$ into $\frac{12}{6} = 2$ clusters $\mathcal{S}_1$ and $\mathcal{S}_2$, each containing six sessions, so that $\mathcal{S}_1 = \{S_1, \ldots, S_6\}$ and $\mathcal{S}_2 = \{S_7, \ldots, S_{12}\}$. Fix $p = 10$, so that $b_1 = 0$ and $p_1 = 2^{10} = 1024$.

For the basis case where $r = 1$, which corresponds to the first epoch, $\mathcal{I}_1^{(1)} = \mathcal{S}_1$, and

$$\begin{aligned}\mathsf{ElevSched}_1^{(1)} &= \mathsf{ElevSched} \mid \mathcal{S}_1 \\ &= 1, 2, \ldots, 6, 6, 5, \ldots, 1, \ldots, 1, 2, \ldots, 6, 6, 5, \ldots, 1, \ldots .\end{aligned}$$

Thus, $\mathsf{ElevSched}_1^{(1)} \uparrow \mathcal{S}_1^{(1)} = 1, 2, \ldots, 6$, so that $i_f^{(1)} = 1$ and $i_l^{(1)} = 6$. We continue to describe the construction of the network $G_1^{(1)}$:

- sessions 1 and 6 traverse edge $e_6^{(1)}$ with $c(e_6^{(1)}) = 2b_1 + \frac{p_1}{2} = 512$;
- for each $i \in \{2, 3, 4, 5\}$, sessions 1 and $i$ traverse $e_i^{(1)}$ with $c(e_i^{(1)}) = 2b_1 + p_1 = 1024$.

The construction of the network $G_2^{(1)}$ is similar; it can be found in Figure 1. We proceed to the case $r = 2$, corresponding to the second epoch, where $\mathcal{I}_1^{(2)} = \mathcal{I}_1^{(1)} \setminus \{1, 6\} = \{2, 3, 4, 5\}$ and

$$\begin{aligned}\mathsf{ElevSched}_1^{(2)} &= \left(\mathsf{ElevSched}_1^{(1)} \downarrow \mathcal{I}_1^{(1)}\right) \mid \mathcal{I}_1^{(2)} \\ &= 5, 4, 3, 2, 2, 3, 4, 5, \ldots, 5, 4, 3, 2, 2, 3, 4, 5, \ldots .\end{aligned}$$

Thus, $\mathsf{ElevSched}_1^{(2)} \uparrow \mathcal{I}_1^{(2)} = 5, 4, 3, 2$, so that $i_f^{(2)} = 5$ and $i_l^{(2)} = 2$. We continue to describe the construction of $G_1^{(2)}$:

- sessions 5 and 2 traverse edge $e_2^{(2)}$ with $c(e_2^{(2)}) = 2b_2 + \frac{p_2}{2} = 1152$;

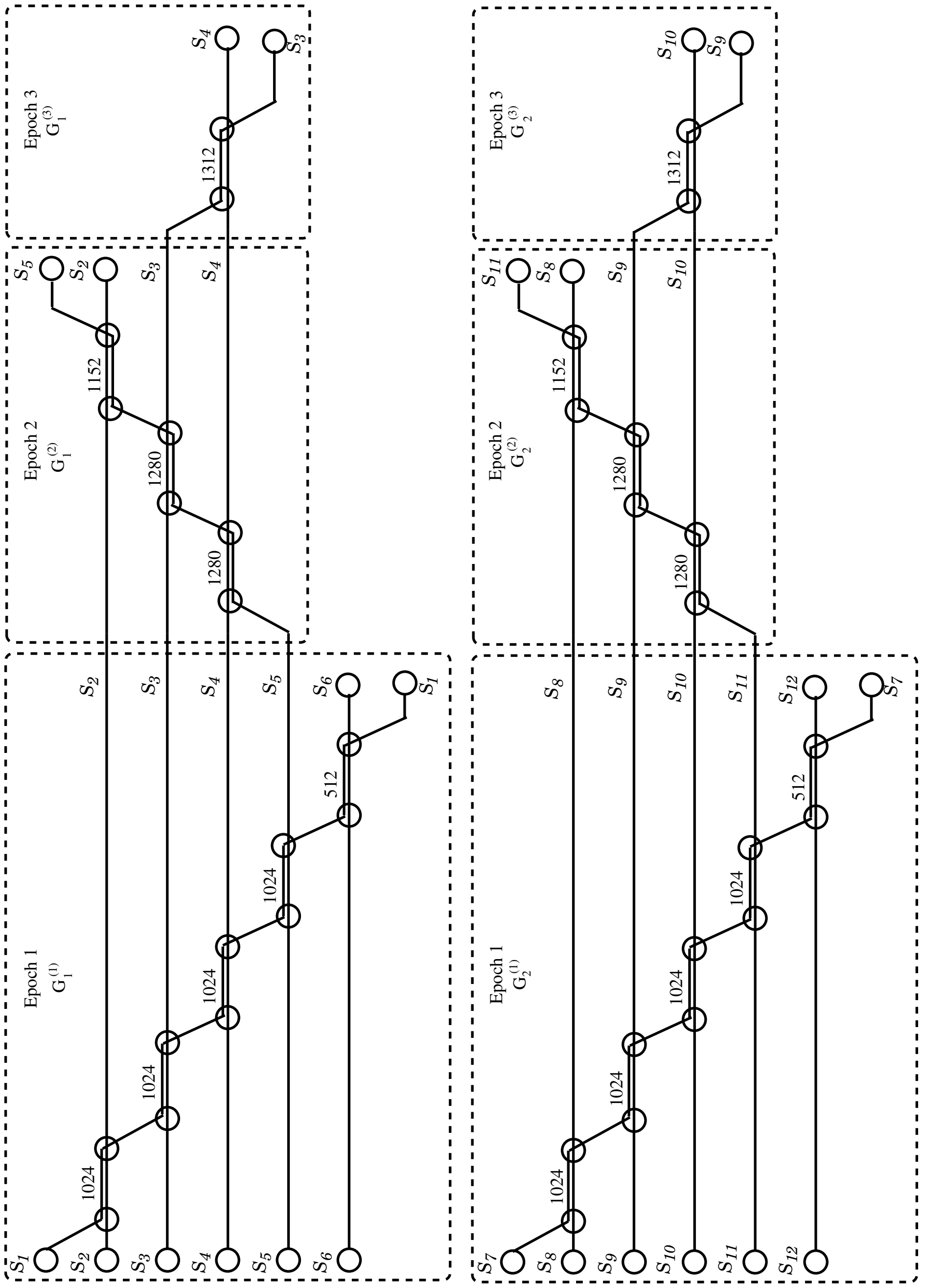


FIG. 1. *The network $G$(ElevSched).*

- for each $i \in \{3,4\}$, sessions 5 and $i$ traverse edge $e_i^{(2)}$ with $c(e_i^{(2)}) = 2b_2 + p_2 =$ 1280.

The construction of the network $G_2^{(2)}$ is similar; it can be found in Figure 1. We proceed to the case $r = 3$, corresponding to the third epoch, where $\mathcal{I}_1^{(3)} = \mathcal{I}_1^{(2)} \setminus$

$\{2,5\} = \{3,4\}$ and

$$\begin{aligned}\mathsf{ElevSched}_1^{(3)} &= \left(\mathsf{ElevSched}_1^{(2)} \downarrow \mathcal{I}_1^{(2)}\right) \mid \mathcal{I}_1^{(3)} \\ &= 3,4,4,3,\ldots,3,4,4,3,\ldots.\end{aligned}$$

Thus, $\mathsf{ElevSched}_1^{(3)} \uparrow \mathcal{I}_1^{(3)} = 3,4$, so that $i_f^{(3)} = 3$ and $i_l^{(3)} = 4$. We continue to describe the construction of $G_1^{(3)}$:

• sessions 3 and 4 traverse edge $e_4^{(3)}$ with $c(e_4^{(3)}) = 2b_3 + \frac{p_3}{2} = 2\cdot 640 + \frac{64}{2} = 1312$.

The construction of the network $G_2^{(3)}$ is similar; it can be found in Figure 1, which also depicts the complete network $G(\mathsf{ElevSched})$. $\square$

## 7. Lower bounds.

**7.1. Oblivious algorithms.** We present a lower bound of $\Omega(dn)$ on the convergence complexity of any optimistic, oblivious, and bottleneck algorithm $\mathsf{Alg} = \langle\mathsf{Sched}, \mathsf{Term}\rangle$ that computes the max-min fair rate vector. The proof uses the network $G\,(\mathsf{Sched})$ constructed in section 6. We start with two immediate technical lemmas that quantify $\Delta_Q(i,e)$ in case edge $e$ is traversed by only two sessions that remain active in state $Q$. (These lemmas will be used for Proposition 7.3.)

LEMMA 7.1. *For an edge e traversed only by sessions $i, i' \in \mathcal{A}_Q \mid e$, $\Delta_Q(i,e) = c(e) - r_Q(i) - \min\{\frac{c(e)}{2}, r_Q(i')\}$.*

Since in the setting of Claim 7.1, $c(e) - r_Q(i) - r_Q\,(i') = resid_Q(e)$, Lemma 7.2 follows.

LEMMA 7.2. *For an edge e traversed only by sessions $i, i' \in \mathcal{A}_Q \mid e$, $\Delta_Q(i,e) \geq resid_Q(e)$.*

We restrict our attention to the execution $\alpha$ of $\mathsf{Alg}$ on the network $G(\mathsf{Sched} \mid \mathcal{S}_j)$ with any particular cluster $\mathcal{S}_j$; for notational simplicity, we shall abuse notation and use $G\,(\mathsf{Sched})$ to denote $G(\mathsf{Sched} \mid \mathcal{S}_j)$ and $\mathcal{S}$ to denote $\mathcal{S}_j$.

For an execution $\alpha$, for any indices $l_1$ and $l_2$, $0 \leq l_1 \leq l_2$, define the set $\mathcal{S} \mid_\alpha (Q_{l_1}, Q_{l_2}] \subseteq \mathcal{S}$ to be $\mathcal{S} \mid_\alpha (Q_{l_1}, Q_{l_2}] = \left\{i_l \mid l_1 < l \leq l_2 \text{ and } i_l \in \mathcal{A}_{Q_{l_1}}\right\}$; roughly speaking, $\mathcal{S} \mid_\alpha (Q_{l_1}, Q_{l_2}]$ contains indices of all sessions active in $Q_{l_1}$ that are scheduled in front of any state following $Q_{l_1}$ and up to and including $Q_{l_2}$. Notice that if $l_1 = l_2$, $\mathcal{S} \mid_\alpha (Q_{l_1}, Q_{l_2}] = \emptyset$. For an execution $\alpha$, for any index $l_1 \geq 0$, define the set $\mathcal{S} \mid_\alpha (Q_{l_1}, \infty) \subseteq \mathcal{S}$ to be $\mathcal{S} \mid_\alpha (Q_{l_1}, \infty) = \left\{i_l \mid l_1 < l \text{ and } i_l \in \mathcal{A}_{Q_{l_1}}\right\}$; roughly speaking, $\mathcal{S} \mid_\alpha (Q_{l_1}, \infty)$ contains indices of all sessions active in $Q_{l_1}$ that are scheduled in front of any state following $Q_{l_1}$. Define $\mathcal{S} \mid_\alpha (Q_{l_1}, Q_{l_2}] \mid e$ and $\mathcal{S} \mid_\alpha (Q_{l_1}, \infty) \mid e$ in the natural way. Recall that for any state $Q_l$, $\widehat{l}$ is the least integer $l' \geq l$ such that $\mathcal{S} \mid_\alpha (Q_l, Q_{l'}] = \mathcal{A}_{Q_l}$, or infinite if no such integer exists.

Define inductively the index sequence $l_0, l_1, \ldots$ as follows. For the basis case, $l_0 = 0$. Assume inductively that for any integer $r \geq 1$, we have defined $l_1, \ldots, l_{r-1}$. For the induction step, define $l_r = \widehat{l_{r-1}}$. Define also the execution fragments $\alpha^{(1)}, \alpha^{(2)}, \ldots$, where for any integer $r \geq 1$, $\alpha^{(r)} = Q_{l_{r-1}}, \ldots, i_{l_r}, Q_{l_r}$. Call each $\alpha^{(r)}$, $r \geq 1$, an *execution epoch in* $\alpha$. Note that in case $l_r = \infty$, for any integer $r \geq 1$, $\alpha^{(r)}$ is the infinite suffix of $\alpha$ following state $Q_{l_{r-1}}$. Thus, write $\alpha = \alpha^{(1)} \cdot \alpha^{(2)} \ldots$. We remark that in case $\alpha$ is infinite, the number of execution epochs in $\alpha$ can still be finite if (and only if) there exists some integer $r \geq 0$ such that $l_r = \infty$. To simplify notation, denote each state $Q_{l_r}$, $r \geq 0$, as $Q^{(r)}$. Thus, $Q^{(r)}$ is the latest state in execution epoch $\alpha^{(r)}$ of $\alpha$. (Note that $Q^{(r)}$ exists if and only if $l_r < \infty$.)

The backbone of our analysis is a technical proposition (Proposition 7.3) that describes the states of execution $\alpha$. The first part of Proposition 7.3 (part (A)) deals with each state starting from $Q^{(r-1)}$, $1 \leq r \leq \frac{d}{2}$, such that not all active sessions in $\overrightarrow{Q^{(r-1)}}$ have yet been scheduled until this state. Thus, $Q^{(r)}$ would be the state immediately following this sequence of states; we will later show that $Q^{(r)}$ is well defined. Observe that the states considered in part (A) for any particular integer $r$, $1 \leq r \leq \frac{d}{2}$, are precisely the states in execution epoch $\alpha^{(r)}$ excluding state $Q^{(r)}$. Part (B) explores properties of state $Q^{(r)}$.

PROPOSITION 7.3 (properties of execution $\alpha$). *For each integer $r$, $1 \leq r \leq \frac{d}{2}$, the following hold for states in execution epoch $\alpha^{(r)}$:*

(A) (properties of states from $Q^{(r-1)}$ to $\overleftarrow{Q^{(r)}}$) *Consider any state $Q$ such that $Q^{(r-1)} \overset{\alpha}{\longmapsto} Q$ and $\mathcal{A}_{\overrightarrow{Q^{(r-1)}}} \not\subseteq \mathcal{S} \mid_\alpha \left(Q^{(r-1)}, Q\right]$. Then the following conditions hold:*

(1) *for each session $i \in \mathcal{A}_{\overrightarrow{Q^{(r-1)}}}$,*

$$r_Q(i) = \begin{cases} b_r + \frac{p_r}{2} & \text{if } i \in \mathcal{S} \mid_\alpha \left(Q^{(r-1)}, Q\right], \\ b_{r-1} + \frac{p_{r-1}}{2} & \text{otherwise}; \end{cases}$$

(2) *for each session $i \in \mathcal{A}_{\overrightarrow{Q^{(r-1)}}}$,*

$$FS_Q(e_i^{(r)}) = \begin{cases} b_r + \frac{p_r}{2} & \text{if } i \notin \left\{i_f^{(r)}, i_l^{(r)}\right\}, \\ b_r + \frac{p_r}{4} & \text{if } i = i_l^{(r)}; \end{cases}$$

(3) *for the edges of $G(\mathsf{Sched})$, it holds, for $Q \neq Q^{(r-1)}$, that*

(a) *edge $e_{i_l^{(r)}}^{(r)}$ is a bottleneck edge for state $Q$;*

(b) *for each session $i \in \mathcal{A}_{\overrightarrow{Q^{(r-1)}}} \setminus \{i_f^{(r)}, i_l^{(r)}\}$,*

$$FS_Q(e_i^{(r)}) \neq MFS_Q(i_f^{(r)}),$$

*so that edge $e_i^{(r)}$ is not a bottleneck edge for state $Q$;*

(c) *for any integer $r'$, $1 \leq r' < r$, and for any edge $e^{(r')} \in E^{(r')}(\mathsf{Sched})$ traversed by session $i \in \mathcal{A}_{\overrightarrow{Q^{(r-1)}}}$,*

$$FS_Q(e^{(r')}) \neq MFS_Q(i),$$

*so that edge $e^{(r')}$ is not a bottleneck edge for state $Q$;*

(d) *for any integer $r'$, $r < r' \leq \frac{d}{2}$, and for any edge $e^{(r')} \in E^{(r')}(\mathsf{Sched})$ traversed by session $i \in \mathcal{A}_{\overrightarrow{Q^{(r-1)}}} \setminus \{i_f^{(r)}, i_l^{(r)}\}$,*

$$FS_Q(e^{(r')}) \neq MFS_Q(i),$$

*so that $e^{(r')}$ is not a bottleneck edge for state $Q$;*

(4) *for each session $i \in \mathcal{A}_{\overrightarrow{Q^{(r-1)}}}$, $i \notin \mathsf{Term}(Q)$;*

(5) *for each session $i \in \mathcal{A}_{\overrightarrow{Q^{(r-1)}}} \setminus \mathcal{S} \mid_\alpha \left(Q^{(r-1)}, Q\right]$ scheduled in front of state $\overrightarrow{Q}$,*

$$\Delta_Q(i) = \begin{cases} \frac{p_r}{2} & \text{if } i \neq i_l^{(r)}, \\ \frac{p_r}{4} & \text{otherwise.} \end{cases}$$

(B) (properties of state $Q^{(r)}$) *The following conditions hold for state* $Q^{(r)}$*:*

(1) $l_r < \infty$*;*

(2) *for each session* $i \in \mathcal{A}_{Q^{\overrightarrow{(r-1)}}}$*,*

$$r_{Q^{(r)}}(i) = \begin{cases} b_r + \frac{p_r}{4} & \text{if } i \in \left\{i_f^{(r)}, i_l^{(r)}\right\}, \\ b_r + \frac{p_r}{2} & \text{if } i \in \mathcal{A}_{Q^{\overrightarrow{(r-1)}}} \setminus \left\{i_f^{(r)}, i_l^{(r)}\right\}; \end{cases}$$

(3) *for each session* $i \in \mathcal{A}_{Q^{\overrightarrow{(r-1)}}}$*,* $i \in \{i_f^{(r)}, i_l^{(r)}\}$ *if and only if* $i \in \mathsf{Term}(Q^{(r)})$*;*

(4) *for each session* $i \in \mathcal{A}_{Q^{\overrightarrow{(r-1)}}} \setminus \{i_f^{(r)}, i_l^{(r)}\}$ *and for any integer* $r'$*,* $1 \le r' \le r$*, for any edge* $e_i^{(r')}$*,*

(a) $resid_{Q^{(r)}}(e_i^{(r')}) \ge \frac{p_r}{4}$*;*

(b) $FS_{\overrightarrow{Q^{(r)}}}(e_i^{(r')}) \ge b_r + \frac{3\,p_r}{4}$.

Proposition 7.3 deals mainly with sessions active in state $Q^{\overrightarrow{(r-1)}}$ for any index $r$, $1 \le r \le \frac{d}{2}$. We start with an informal description of the conditions in part (A). Condition (A/1) determines rates of active sessions in state $Q$. Condition (A/2) determines the fair shares of all edges in epoch $r$; condition (A/3) establishes that edge $e_{i_l^{(r)}}^{(r)}$ is the only bottleneck edge for state $Q$. Condition (A/4) guarantees that no session is terminated in state $Q$. Finally, condition (A/5) determines the increase in state $Q$ for sessions active in $Q^{\overrightarrow{(r-1)}}$ that are not yet scheduled. We now continue with the conditions in part (B). Condition (B/1) asserts that all active sessions in state $Q^{\overrightarrow{(r-1)}}$ must be scheduled in execution epoch $\alpha^{(r)}$; condition (B/2) specifies their rates upon completion of $\alpha^{(r)}$. Moreover, condition (B/3) determines which of these sessions are terminated upon completion of $\alpha^{(r)}$. Condition (B/4) provides lower bounds on the residual capacity and fair share of some edges from previous epochs upon completion of $\alpha^{(r)}$.

We note that conditions (B/1) and (B/3) will suffice by themselves to imply the lower bound. However, the rather technical remaining conditions are necessary to assume inductively in the proof of conditions (B/1) and (B/3). To simplify notation, we will denote $e_{i_l^{(r)}}^{(r)}$ as $e_l^{(r)}$.

*Proof.* The proof is by induction on $r$. For the sake of shortening the proof, we merge the proof for the basis case (where $r = 1$) and the proof for the induction step; thus, the case $r = 1$ will be treated separately (where needed) along the proof of the induction step.

We assume, as our induction hypothesis, that the claims hold for all integers less than some fixed integer $r$, $1 \le r \le \frac{d}{2}$. Notice that if $r = 1$, the induction hypothesis is empty. We proceed to the induction step, where we prove the claims for $r$.

*Proof of part* (A). The proof is by induction on $Q$. For the sake of shortening the proof, we again merge the proof for the basis case (where $Q = Q^{(r-1)}$) and the proof for the induction step; thus, the case $Q = Q^{(r-1)}$ will be treated separately (where needed) along the proof for the induction step (on states).

Fix any state $Q$ such that $Q^{(r-1)} \stackrel{\alpha}{\longmapsto} Q$ and $\mathcal{A}_{Q^{\overrightarrow{(r-1)}}} \not\subseteq \mathcal{S} \mid_{\alpha} \left(Q^{(r-1)}, Q\right]$, and assume that for each state $Q'$ such that $Q^{(r-1)} \stackrel{\alpha}{\longmapsto} Q' \stackrel{\alpha}{\longrightarrow} Q$, the claims of part (A) hold for $Q'$; thus, we assume, as our induction hypothesis, that the claims of part (A) hold for all states from $Q^{(r-1)}$ through but not including state $Q$. Notice that if $Q = Q^{(r-1)}$, the induction hypothesis is empty. We now proceed with the induction step, where we prove the claims for $Q$.

*Proof of* (A/1). There are two cases to consider.

(1) Assume first that $Q = Q^{(r-1)}$. Then $\mathcal{S} \mid_\alpha \left(Q^{(r-1)}, Q\right] = \emptyset$. In case $r = 1$, $Q = Q_0$, and condition (A/1) holds trivially since all session rates are initially zero and $b_0 = p_0 = 0$. So assume $r > 1$. By the induction hypothesis of induction on $r$ (condition (B/3)), $\mathcal{A}_{\overrightarrow{Q^{(r-1)}}} = \mathcal{A}_{\overrightarrow{Q^{(r-2)}}} \setminus \{i_f^{(r-1)}, i_l^{(r-1)}\}$. Hence, condition (A/1) follows from the induction hypothesis of induction on $r$ (condition (B/2)).

(2) Now assume that $Q \neq Q^{(r-1)}$. We proceed by case analysis on $i_Q$ (the session scheduled in front of state $Q$).

(a) Assume first that $i_Q \notin \mathcal{A}_{\overrightarrow{Q^{(r-1)}}}$. (Notice that this case need not be considered when $r = 1$, since all sessions are active in $\overrightarrow{Q^{(0)}} = Q_1$.) Then $\mathcal{S} \mid_\alpha \left(Q^{(r-1)}, Q\right] = \mathcal{S} \mid_\alpha (Q^{(r-1)}, \overleftarrow{Q}]$. Since, in addition, no session rates change from $\overleftarrow{Q}$ to $Q$, the claim follows inductively. So we proceed to the cases where $i_Q \in \mathcal{A}_{\overrightarrow{Q^{(r-1)}}}$.

(b) Next assume that $i_Q = i_f^{(r)}$. There are two subcases to consider.

(i) First, take $i_f^{(r)} \notin \mathcal{S} \mid_\alpha (Q^{(r-1)}, \overleftarrow{Q}]$; that is, $i_f^{(r)}$ has not been scheduled in front of any state between $Q^{(r-1)}$ and $Q$. Recall that $i_f^{(r)}$ is the session scheduled first (following state $Q^{(r-1)}$) among active sessions in $\overrightarrow{Q^{(r-1)}}$; so, for each session $i \in \mathcal{A}_{\overrightarrow{Q^{(r-1)}}}$, $i \notin \mathcal{S} \mid_\alpha (Q^{(r-1)}, Q]$. Thus, by the induction hypothesis of induction on states (condition (A/1)), $r_{\overleftarrow{Q}}(i) = b_{r-1} + \frac{p_{r-1}}{2} = b_r$ (by recursive definition of $b_r$). In particular, $r_{\overleftarrow{Q}}(i_f^{(r)}) = b_r$. By the induction hypothesis of induction on states (condition (A/5)), $\Delta_{\overleftarrow{Q}}(i_f^{(r)}) = \frac{p_r}{2}$. Thus, by the `update` operation, $r_Q(i_f^{(r)}) = r_{\overleftarrow{Q}}(i_f^{(r)}) + \Delta_{\overleftarrow{Q}}(i_f^{(r)}) = b_r + \frac{p_r}{2}$.
Since $p_r \neq 0$, it follows that for each session $i \in \mathcal{A}_{\overrightarrow{Q^{(r-1)}}} \setminus \{i_f^{(r)}\}$, $r_{\overleftarrow{Q}}(i) < r_Q(i_f^{(r)})$. Thus, by the `update` operation, $r_Q(i) = r_{\overleftarrow{Q}}(i) = b_{r-1} + \frac{p_{r-1}}{2}$. Since $\mathcal{S} \mid_\alpha (Q^{(r-1)}, Q] = \{i_f^{(r)}\}$, it follows that for each session $i \in \mathcal{A}_{\overrightarrow{Q^{(r-1)}}}$,

$$r_Q(i) = \begin{cases} b_r + \frac{p_r}{2} & \text{if } i \in \mathcal{S} \mid_\alpha \left(Q^{(r-1)}, Q\right], \\ b_{r-1} + \frac{p_{r-1}}{2} & \text{otherwise.} \end{cases}$$

(ii) Now take $i_f^{(r)} \in \mathcal{S} \mid_\alpha (Q^{(r-1)}, \overleftarrow{Q}]$. Then $\mathcal{S} \mid_\alpha (Q^{(r-1)}, Q] = \mathcal{S} \mid_\alpha (Q^{(r-1)}, \overleftarrow{Q}]$. Clearly, $i_l^{(r)} \notin \mathcal{S} \mid_\alpha (Q^{(r-1)}, \overleftarrow{Q}]$, while $i_f^{(r)} \in \mathcal{S} \mid_\alpha (Q^{(r-1)}, \overleftarrow{Q}]$. By construction of $G(\mathsf{Sched})$, session $i_f^{(r)}$ traverses edge $e_l^{(r)}$ with $c(e_l^{(r)}) = 2b_r + \frac{p_r}{2}$, as does session $i_l^{(r)}$. By the induction hypothesis of induction on states (condition (A/1)), $r_{\overleftarrow{Q}}(i_f^{(r)}) = b_r + \frac{p_r}{2}$, while $r_{\overleftarrow{Q}}(i_l^{(r)}) = b_{r-1} + \frac{p_{r-1}}{2} = b_r$. By Lemma 7.1,

$$\begin{aligned} &\Delta_{\overleftarrow{Q}}(i_f^{(r)}, e_l^{(r)}) \\ &= c(e_l^{(r)}) - r_{\overleftarrow{Q}}(i_f^{(r)}) - \min\left\{\frac{c(e_l^{(r)})}{2}, r_{\overleftarrow{Q}}(i_l^{(r)})\right\} \\ &= 2\,b_r + \frac{p_r}{2} - \left(b_r + \frac{p_r}{2}\right) - \min\left\{b_r + \frac{p_r}{2}, b_r\right\} \\ &= 0, \end{aligned}$$

so that $\Delta_{\overleftarrow{Q}}(i_f^{(r)}) = 0$. Thus, by the `update` operation, for each session $i \in \mathcal{A}_{\overleftarrow{Q}}$, $r_Q(i) = r_{\overleftarrow{Q}}(i)$. By the induction hypothesis of induction on states (condition (A/4)), it follows that $\mathcal{A}_{\overleftarrow{Q}} = \mathcal{A}_{\overrightarrow{Q^{(r-1)}}}$. Hence, the induction hypothesis of induction on states (condition (A/1)) implies that for each session $i \in \mathcal{A}_{\overrightarrow{Q^{(r-1)}}}$,

$$r_Q(i) = \begin{cases} b_r + \frac{p_r}{2} & \text{if } i \in \mathcal{S}\mid_\alpha \left(Q^{(r-1)}, Q\right], \\ b_{r-1} + \frac{p_{r-1}}{2} & \text{otherwise.} \end{cases}$$

(c) Assume now that $i_Q \neq i_f^{(r)}$. There are two subcases to consider.

(i) First consider the case where $i_Q \notin \mathcal{S}\mid_\alpha (Q^{(r-1)}, \overleftarrow{Q}]$. By the induction hypothesis of induction on states (condition (A/1)), it follows that $r_{\overleftarrow{Q}}(i_Q) = b_{r-1} + \frac{p_{r-1}}{2} = b_r$; moreover, by condition (A/5), $\Delta_{\overleftarrow{Q}}(i_Q) = \frac{p_r}{2}$. Thus, by the `update` operation, $r_Q(i_Q) = r_{\overleftarrow{Q}}(i_Q) + \Delta_{\overleftarrow{Q}}(i_Q) = b_r + \frac{p_r}{2}$.
By induction hypothesis of induction on states (condition (A/1)), it follows that for each session $i \in \mathcal{A}_{\overrightarrow{Q^{(r-1)}}} \setminus \{i_Q\}$, $r_{\overleftarrow{Q}}(i) \leq b_r + \frac{p_r}{2}$; hence, $r_{\overleftarrow{Q}}(i) \leq r_Q(i_Q)$. Thus, by the `update` operation, $r_Q(i) = r_{\overleftarrow{Q}}(i)$. Since $\mathcal{S}\mid_\alpha (Q^{(r-1)}, Q] = \mathcal{S}\mid_\alpha (Q^{(r-1)}, \overleftarrow{Q}] \bigcup \{i_Q\}$, the induction hypothesis of induction on states (condition (A/1)) implies now that for each session $i \in \mathcal{A}_{\overrightarrow{Q^{(r-1)}}}$,

$$r_Q(i) = \begin{cases} b_r + \frac{p_r}{2} & \text{if } i \in \mathcal{S}\mid_\alpha \left(Q^{(r-1)}, Q\right], \\ b_{r-1} + \frac{p_{r-1}}{2} & \text{otherwise.} \end{cases}$$

(ii) Finally, consider the case where $i_Q \in \mathcal{S}\mid_\alpha (Q^{(r-1)}, \overleftarrow{Q}] \setminus \{i_f^{(r)}\}$. Then $\mathcal{S}\mid_\alpha (Q^{(r-1)}, Q] = \mathcal{S}\mid_\alpha (Q^{(r-1)}, \overleftarrow{Q}]$. Clearly, both $i_Q, i_f^{(r)} \in \mathcal{S}\mid_\alpha (Q^{(r-1)}, \overleftarrow{Q}]$. Thus, by the induction hypothesis of induction on states (condition (A/1)), $r_{\overleftarrow{Q}}(i_Q) = r_{\overleftarrow{Q}}(i_f^{(r)}) = b_r + \frac{p_r}{2}$. By construction of $G(\mathsf{Sched})$, $c(e_{i_Q}^{(r)}) = 2\,b_r + p_r$. It follows that $r_{\overleftarrow{Q}}(i_Q) = r_{\overleftarrow{Q}}(i_f^{(r)}) = \frac{c(e_{i_Q}^{(r)})}{2}$. Hence, Lemma 7.1 implies that $\Delta_{\overleftarrow{Q}}(i_Q, e_{i_Q}^{(r)}) = 0$, so that $\Delta_{\overleftarrow{Q}}(i_Q) = 0$. Thus, by the `update` operation, for each session $i \in \mathcal{A}_{\overleftarrow{Q}}$, $r_Q(i) = r_{\overleftarrow{Q}}(i)$. By the induction hypothesis of induction on states (condition (A/4)), it follows that $\mathcal{A}_{\overleftarrow{Q}} = \mathcal{A}_{\overrightarrow{Q^{(r-1)}}}$. Hence, the induction hypothesis of induction on states (condition (A/1)) implies that for each session $i \in \mathcal{A}_{\overrightarrow{Q^{(r-1)}}}$,

$$r_Q(i) = \begin{cases} b_r + \frac{p_r}{2} & \text{if } i \in \mathcal{S}\mid_\alpha \left(Q^{(r-1)}, Q\right], \\ b_{r-1} + \frac{p_{r-1}}{2} & \text{otherwise.} \quad \square \end{cases}$$

*Proof of* (A/2). First consider edge $e_i^{(r)}$ for any session $i \in \mathcal{A}_{\overrightarrow{Q^{(r-1)}}} \setminus \{i_f^{(r)}, i_l^{(r)}\}$, with capacity $c(e_i^{(r)}) = 2\,b_r + p_r$, which is traversed by sessions $i_f^{(r)}$ and $i$. Either by definition of $Q_0$ in case $Q = Q^{(r-1)}$ and $r = 1$, or by the induction hypothesis of induction on $r$ (condition (B/3)) in case $Q = Q^{(r-1)}$ and $r \neq 1$, or by the induction hypothesis of induction on states (condition (A/4)) in case $Q \neq Q^{(r-1)}$, it follows that both $i_f^{(r)}, i \in \mathcal{A}_Q$, so that $\mathcal{A}_Q \mid e_i^{(r)} = \{i_f^{(r)}, i\}$. Hence,

$$\begin{aligned} FS_Q(e_i^{(r)}) &= \frac{c(e_i^{(r)}) - \mathit{allot}_Q(e_i^{(r)})}{|\mathcal{A}_Q \mid e_i^{(r)}|} \\ &= \frac{c(e_i^{(r)})}{2} \\ &= b_r + \frac{p_r}{2}. \end{aligned}$$

Now consider edge $e_l^{(r)}$ with capacity $c(e_l^{(r)}) = 2b_r + \frac{p_r}{2}$, which is traversed by sessions $i_f^{(r)}$ and $i_l^{(r)}$. Either by definition of $Q_0$ in case $Q = Q^{(r-1)}$ and $r = 1$, or by the induction hypothesis of induction on $r$ (condition (B/3)) in case $Q = Q^{(r-1)}$ and $r \neq 1$, or by the induction hypothesis of induction on states (condition (A/4)) in case $Q \neq Q^{(r-1)}$, it follows that both $i_f^{(r)}, i_l^{(r)} \in \mathcal{A}_Q$, so that $\mathcal{A}_Q \mid e_l^{(r)} = \{i_f^{(r)}, i_l^{(r)}\}$. Hence,

$$\begin{aligned} FS_Q(e_l^{(r)}) &= \frac{c(e_l^{(r)}) - \mathit{allot}_Q(e_l^{(r)})}{\left|\mathcal{A}_Q \mid e_l^{(r)}\right|} \\ &= \frac{c(e_l^{(r)})}{2} \\ &= b_r + \frac{p_r}{4}. \qquad \Box \end{aligned}$$

*Proof of* (A/3/a). By construction of $G(\mathsf{Sched})$, edge $e_l^{(r)}$ is traversed by sessions $i_f^{(r)}$ and $i_l^{(r)}$. We prove that each of them receives its minimum fair share on edge $e_l^{(r)}$.

(1) First, take session $i_f^{(r)}$, which traverses the following edges.
- edge $e_l^{(r)}$; by condition (A/2) shown above, $FS_Q(e_l^{(r)}) = b_r + \frac{p_r}{4}$;
- edge $e_i^{(r)}$ for each session $i \in \mathcal{A}_{\overrightarrow{Q^{(r-1)}}} \setminus \{i_f^{(r)}, i_l^{(r)}\}$; by condition (A/2) shown above, $FS_Q(e_i^{(r)}) = b_r + \frac{p_r}{2}$;
- edge $e^{(r')} \in E^{(r')}(\mathsf{Sched})$ for each integer $r'$, $1 \leq r' < r$, in case $r > 1$. By the induction hypothesis of induction on $r$ (condition (B/4/b)), $FS_{\overrightarrow{Q^{(r-1)}}}(e^{(r')}) \geq b_{r-1} + \frac{3\,p_{r-1}}{4} = b_r + p_r$ (by recursive definition of $b_r$ and $p_r$). Since $Q \neq Q^{(r-1)}$, the induction hypothesis of induction on states (condition (A/4)) is nonempty and implies that $\mathcal{A}_Q = \mathcal{A}_{\overrightarrow{Q^{(r-1)}}}$. It follows that $FS_Q(e^{(r')}) = FS_{\overrightarrow{Q^{(r-1)}}}(e^{(r')})$, so that $FS_Q(e^{(r')}) \geq b_r + p_r$.

Hence, $MFS_Q(i_f^{(r)}) = b_r + \frac{p_r}{4}$, so that $MFS_Q(i_f^{(r)}) = FS_Q(e_l^{(r)})$.

(2) Now take session $i_l^{(r)}$, which traverses the following edges.
- edge $e_l^{(r)}$; by condition (A/2) shown above, $FS_Q(e_l^{(r)}) = b_r + \frac{p_r}{4}$;
- edge $e^{(r')} \in E^{(r')}(\mathsf{Sched})$ for each integer $r'$, $1 \leq r' < r$, in case $r > 1$. By the induction hypothesis of induction on $r$ (condition (B/4/b)), $FS_{\overrightarrow{Q^{(r-1)}}}(e^{(r')}) \geq b_{r-1} + \frac{3\,p_{r-1}}{4} = b_r + p_r$. Since $Q \neq Q^{(r-1)}$, the induction hypothesis of induction on states (condition (A/4)) implies that

$\mathcal{A}_Q = \mathcal{A}_{\overrightarrow{Q^{(r-1)}}}$. It follows that $FS_Q(e^{(r')}) = FS_{\overrightarrow{Q^{(r-1)}}}(e^{(r')})$, so that $FS_Q(e^{(r')}) \geq b_r + p_r$.

These imply that $MFS_Q(i_l^{(r)}) = b_r + \frac{p_r}{4}$, so that $MFS_Q(i_l^{(r)}) = FS_Q(e_l^{(r)})$. Hence $MFS_Q(i_f^{(r)}) = MFS_Q(i_l^{(r)}) = FS_Q(e_l^{(r)})$, so that $e_l^{(r)}$ is a bottleneck edge for $Q$. ◻

*Proof of* (A/3/b). Consider edge $e_i^{(r)}$ for any session $i \in \mathcal{A}_{\overrightarrow{Q^{(r-1)}}} \setminus \{i_f^{(r)}, i_l^{(r)}\}$, which is traversed by sessions $i_f^{(r)}$ and $i$. By condition (A/3/a) shown above, edge $e_l^{(r)}$, traversed by session $i_f^{(r)}$, is a bottleneck edge for state $Q$, so that $MFS_Q(i_f^{(r)}) = FS_Q(e_l^{(r)})$. By condition (A/2) shown above, $FS_Q(e_l^{(r)}) = b_r + \frac{p_r}{4}$ and $FS_Q(e_i^{(r)}) = b_r + \frac{p_r}{2}$. Since $p_r \neq 0$, it follows that $FS_Q(e_l^{(r)}) < FS_Q(e_i^{(r)})$, so that $MFS_Q(i_f^{(r)}) < FS_Q(e_i^{(r)})$. It follows that $e_i^{(r)}$ is not a bottleneck edge for $Q$. ◻

*Proof of* (A/3/c). The claim holds vacuously in case $r = 1$. So assume $r > 1$. Consider any edge $e^{(r')} \in E^{(r')}(\mathsf{Sched})$ for any integer $r'$, $1 \leq r' < r$, traversed by some session $i \in \mathcal{A}_{\overrightarrow{Q^{(r-1)}}}$. By the induction hypothesis of induction on $r$ (condition (B/4/b)), $FS_{\overrightarrow{Q^{(r-1)}}}(e^{(r')}) \geq b_{r-1} + \frac{3\,p_{r-1}}{4}$. Since $Q \neq Q^{(r-1)}$, the induction hypothesis of induction on states (condition (A/4)) is nonempty and implies that $\mathcal{A}_Q = \mathcal{A}_{\overrightarrow{Q^{(r-1)}}}$. It follows that $FS_Q(e^{(r')}) = FS_{\overrightarrow{Q^{(r-1)}}}(e^{(r')})$, so that $FS_Q(e^{(r')}) \geq p_{r-1} + \frac{3\,p_{r-1}}{4} = b_r + p_r$. Session $i$ also traverses edge $e_i^{(r)}$. By condition (A/2) shown above, $FS_Q(e_i^{(r)}) \leq b_r + \frac{p_r}{2}$.

Since $p_r \neq 0$, it follows that $FS_Q(e^{(r')}) > FS_Q(e_i^{(r)})$, so that $FS_Q(e^{(r')}) > MFS_Q(i)$. It follows that $e^{(r')}$ is not a bottleneck edge for $Q$. ◻

*Proof of* (A/3/d). Consider any edge $e^{(r')} \in E^{(r')}(\mathsf{Sched})$ for any integer $r'$, $r < r' \leq \frac{d}{2}$. Take any session $i$ traversing $e^{(r')}$; by construction of $G(\mathsf{Sched})$, $i \in \mathcal{A}_{\overrightarrow{Q^{(r-1)}}} \setminus \{i_f^{(r)}, i_l^{(r)}\}$, and $i$ traverses also $e_i^{(r)}$ with $c(e_i^{(r)}) = 2\,b_r + p_r$. By condition (A/2) shown above, $FS_Q(e_i^{(r)}) = b_r + \frac{p_r}{2} = \frac{c(e_i^{(r)})}{2}$. Since $Q \neq Q^{(r-1)}$, the induction hypothesis of induction on states (condition (A/4)) is nonempty and implies that both sessions traversing $e^{(r')}$ are active in $Q$. So, $FS_Q(e^{(r')}) = \frac{c(e^{(r')})}{2}$.

By Lemma 6.1, $c(e_i^{(r)}) < c(e^{(r')})$. So, $FS_Q(e_i^{(r)}) < FS_Q(e^{(r')})$. By definition of minimum fair share, $MFS_Q(i) < FS_Q(e^{(r')})$. Hence, $e^{(r')}$ is not a bottleneck edge for $Q$. ◻

*Proof of* (A/4). If $Q = Q^{(r-1)}$, the claim holds trivially. So assume $Q \neq Q^{(r-1)}$. By condition (A/3/b) shown above, any edge $e_i^{(r)}$ with $i \in \mathcal{A}_{\overrightarrow{Q^{(r-1)}}} \setminus \{i_f^{(r)}, i_l^{(r)}\}$ is not a bottleneck edge for $Q$. Moreover, by conditions (A/3/c) and (A/3/d) shown above, neither is any edge $e^{(r')} \in E^{(r')}(\mathsf{Sched})$ with $1 \leq r' < r$ or $r < r' \leq \frac{d}{2}$. Since $\mathsf{Alg}$ is bottleneck, none of these edges causes the termination of a session in $Q$. Thus, it remains to prove only that edge $e_{i_l^{(r)}}^{(r)}$ does not cause either the termination of any of the sessions $i_f^{(r)}$ and $i_l^{(r)}$ traversing it.

Since $p_r \neq 0$, conditions (A/1) and (A/2) shown above imply that $r_Q(i_f^{(r)}) \neq FS_Q(e_l^{(r)})$ and $r_Q(i_l^{(r)}) \neq FS_Q(e_l^{(r)})$. Since $\mathsf{Alg}$ is bottleneck, it follows that $i_f^{(r)}, i_l^{(r)} \notin \mathsf{Term}\,(Q)$. ◻

*Proof of* (A/5). The proof is obtained by case analysis on $i$.

1. Assume first that $i = i_f^{(r)}$. We first prove by case analysis on $Q$ that, in this case, for each session $k \in \mathcal{A}_{\overrightarrow{Q^{(r-1)}}}$, $r_Q(k) = b_r$. (This will be useful for some

later calculations.)

- First take $Q = Q^{(r-1)}$. In case $r = 1$ where $Q = Q_0$, all session rates are zero in $Q_0$. Since $b_0 = p_0 = 0$, it follows that $r_Q(k) = b_{r-1} + \frac{p_{r-1}}{2}$. Assume now that $r > 1$. Since $k \in \mathcal{A}_{\overrightarrow{Q^{(r-1)}}}$, by construction of $G(\mathsf{Sched})$ and by the induction hypothesis of induction on $r$ (condition (B/3)), $k \notin \{i_f^{(r-1)}, i_l^{(r-1)}\}$. Thus, the induction hypothesis of induction on $r$ (condition (B/2)) implies that $r_Q(k) = b_{r-1} + \frac{p_{r-1}}{2}$.
- Now take $Q \neq Q^{(r-1)}$. Since $i_f^{(r)} \in \mathcal{A}_{\overrightarrow{Q^{(r-1)}}} \setminus \mathcal{S} \mid_\alpha \big(Q^{(r-1)}, Q\big]$, $k \in \mathcal{A}_{\overrightarrow{Q^{(r-1)}}} \setminus \mathcal{S} \mid_\alpha \big(Q^{(r-1)}, Q\big]$ as well. So, by condition (A/1) shown above, $r_Q(k) = b_{r-1} + \frac{p_{r-1}}{2}$.

So, in all cases, $r_Q(k) = b_{r-1} + \frac{p_{r-1}}{2} = b_r$. By construction of $G(\mathsf{Sched})$, $i_f^{(r)}$ traverses the following edges.

- edge $e_l^{(r)}$ with $c(e_l^{(r)}) = 2b_r + \frac{p_r}{2}$;
- edge $e_k^{(r)}$ for each session $k \in \mathcal{A}_{\overrightarrow{Q^{(r-1)}}} \setminus \{i_f^{(r)}, i_l^{(r)}\}$, with $c(e_k^{(r)}) = 2b_r + p_r$;
- edge $e^{(r')} \in E^{(r')}(\mathsf{Sched})$ for each index $r'$, $1 \leq r' < r$, in case $r > 1$.

We next calculate separately the increases for $i_f^{(r)}$ in $Q$ allowed by these edges.

(a) First consider edge $e_l^{(r)}$, which is also traversed by $i_l^{(r)} \in \mathcal{A}_{\overrightarrow{Q^{(r-1)}}}$. By Lemma 7.1,

$$\begin{aligned}
\Delta_Q(i_f^{(r)}, e_l^{(r)}) &= c(e_l^{(r)}) - r_Q(i_f^{(r)}) - \min\left\{\frac{c((e_l^{(r)})}{2}, r_Q(i_l^{(r)})\right\} \\
&= 2b_r + \frac{p_r}{2} - b_r - \min\left\{b_r + \frac{p_r}{4}, b_r\right\} \\
&= \frac{p_r}{2}.
\end{aligned}$$

(b) Next consider any edge $e_k^{(r)}$ for any session $k \in \mathcal{A}_{\overrightarrow{Q^{(r-1)}}} \setminus \{i_f^{(r)}, i_l^{(r)}\}$, which is also traversed by session $k$. By Lemma 7.1,

$$\begin{aligned}
\Delta_Q(i_f^{(r)}, e_k^{(r)}) &= c(e_k^{(r)}) - r_Q(i_f^{(r)}) - \min\{c(e_k^{(r)}), r_Q(k)\} \\
&= 2\,b_r + p_r - b_r - \min\left\{b_r + \frac{p_r}{2}, b_r\right\} \\
&= p_r.
\end{aligned}$$

(c) Finally, consider edge $e^{(r')}$ for any integer $r'$, $1 \leq r' < r$, in case $r > 1$. By the induction hypothesis of induction on $r$ (condition (B/4/a)), $resid_{Q^{(r-1)}}(e^{(r')}) \geq \frac{p_{r-1}}{4}$. Recall that $i_f^{(r)}$ is the session scheduled first (following state $Q^{(r-1)}$) among active sessions in $\overrightarrow{Q^{(r-1)}}$. Since $i_f^{(r)} \in \mathcal{A}_{\overrightarrow{Q^{(r-1)}}} \setminus \mathcal{S} \mid_\alpha \big(Q^{(r-1)}, Q\big]$, it follows that for each session $i \in \mathcal{A}_{\overrightarrow{Q^{(r-1)}}}$, $i \in \mathcal{A}_{\overrightarrow{Q^{(r-1)}}} \setminus \mathcal{S} \mid_\alpha \big(Q^{(r-1)}, Q\big]$. Thus, rates of all sessions are preserved from state $Q^{(r-1)}$ to $Q$. Hence, $resid_Q(e^{(r')}) = resid_{Q^{(r-1)}}(e^{(r')}) \geq \frac{p_{r-1}}{4} = p_r$. By Lemma 7.2, it follows that $\Delta_Q(i_f^{(r)}, e^{(r')}) \geq resid_Q(e^{(r')}) \geq p_r$.

Thus, by definition of increase, it follows that $\Delta_Q(i_f^{(r)}) = \frac{p_r}{2}$.

2. Now assume that $i \notin \{i_f^{(r)}, i_l^{(r)}\}$. Since $i \in \mathcal{A}_{\overrightarrow{Q^{(r-1)}}} \setminus \mathcal{S} \mid_\alpha \big(Q^{(r-1)}, Q\big]$, by condition (A/1) shown above, we have that $r_Q(i) = b_{r-1} + \frac{p_{r-1}}{2}$. By construction of $G(\mathsf{Sched})$ $i$ traverses the following edges.

- edge $e_i^{(r)}$ with $c(e_i^{(r)}) = 2\,b_r + p_r$;
- edge $e^{(r')} \in E^{(r')}(\mathsf{Sched})$ for each integer $r'$, $1 \leq r' < r$, in case $r > 1$;
- edge $e^{(r'')} \in E^{(r'')}(\mathsf{Sched})$ for any integer $r''$, $r < r'' \leq \frac{d}{2}$.

We next calculate separately the increases for $i$ in $Q$ allowed by these edges.

(a) First consider edge $e_i^{(r)}$, which is also traversed by $i_f^{(r)}$. Since $i$ is scheduled in front of state $\overrightarrow{Q}$, it follows that $i_f^{(r)} \in \mathcal{S}\,|_\alpha\,\big(Q^{(r-1)}, Q\big]$. Thus, condition (A/1) shown above implies that $r_Q(i_f^{(r)}) = b_r + \frac{p_r}{2} = \frac{c(e_i^{(r)})}{2}$. By Lemma 7.1,

$$\begin{aligned}\Delta_Q(i, e_i^{(r)}) &= c(e_i^{(r)}) - r_Q(i) - \min\left\{\frac{c(e_i^{(r)})}{2}, r_Q(i_f^{(r)})\right\}\\ &= c(e_i^{(r)}) - r_Q(i) - \frac{c(e_i^{(r)})}{2}\\ &= \frac{c(e_i^{(r)})}{2} - \Big(b_{r-1} + \frac{p_{r-1}}{2}\Big).\end{aligned}$$

Alternatively, we derive that $\Delta_Q(i, e_i^{(r)}) = b_r + \frac{p_r}{2} - (b_{r-1} + \frac{p_{r-1}}{2}) = \frac{p_r}{2}$.[4]

(b) Next consider edge $e_i^{(r')}$ for any integer $r'$, $1 \leq r' < r$, in case $r > 1$, which is also traversed by $i_f^{(r')} \notin \{i_f^{(r)}, i_l^{(r)}\}$. Induction hypothesis of induction on $r$ (condition (B/3)) implies that $i_f^{(r')} \notin \mathcal{A}_{\overrightarrow{Q^{(r-1)}}}$; hence, $i_f^{(r')} \notin \mathcal{A}_Q$. So, $r_{Q^{(r-1)}}(i_f^{(r')}) = r_Q(i_f^{(r')})$. Recall that $r_Q(i) = b_{r-1} + \frac{p_{r-1}}{2}$. On the other hand, by the induction hypothesis of induction on $r$ (condition (B/2)), $r_{Q^{(r-1)}}(i) = b_{r-1} + \frac{p_{r-1}}{2}$. Hence, $r_Q(i) = r_{Q^{(r-1)}}(i)$. It now follows that $\mathit{resid}_Q(e_i^{(r')}) = \mathit{resid}_{Q^{(r-1)}}(e_i^{(r')})$. By the induction hypothesis of induction on $r$ (condition (B/4/a)), $\mathit{resid}_{Q^{(r-1)}}(e_i^{(r')}) \geq \frac{p_{r-1}}{4}$, so that $\mathit{resid}_Q(e_i^{(r')}) \geq \frac{p_{r-1}}{4}$. By Lemma 7.2, $\Delta_Q(i, e_i^{(r')}) \geq \frac{p_{r-1}}{4} = p_r$.

(c) Finally, consider edge $e^{(r'')}$ for any integer $r''$, $r < r'' \leq \frac{d}{2}$, which is also traversed by some other session $k \in \mathcal{A}_{\overrightarrow{Q^{(r-1)}}} \setminus \{i_f^{(r)}, i_l^{(r)}\}$. By condition (A/1) shown above and by recursive definition of $b_r$, $r_Q(k) \leq b_r + \frac{p_r}{2} = \frac{c(e_i^{(r)})}{2}$. By Lemma 6.1, $c(e^{(r'')}) > c(e_i^{(r)})$. By Lemma 7.1,

$$\begin{aligned}\Delta_Q(i, e^{(r'')}) &= c(e^{(r'')}) - r_Q(i) - \min\left\{\frac{c(e^{(r'')})}{2}, r_Q(k)\right\}\\ &\geq c(e^{(r'')}) - r_Q(i) - \min\left\{\frac{c(e^{(r'')})}{2}, \frac{c(e_i^{(r)})}{2}\right\}\\ &\geq c(e^{(r'')}) - r_Q(i) - \frac{c(e^{(r'')})}{2}\\ &= \frac{c(e^{(r'')})}{2} - \Big(b_{r-1} + \frac{p_{r-1}}{2}\Big).\end{aligned}$$

[4]Both expressions provided for $\Delta_Q(i, e_i^{(r)})$ will be needed in the rest of the proof.

Since $c(e^{(r'')}) > c(e_i^{(r)})$, comparing the first expression for $\Delta_Q(i, e_i^{(r)})$ to $\Delta_Q(i, e^{(r'')})$ implies that $\Delta_Q(i, e^{(r'')}) > \Delta_Q(i, e_i^{(r)})$.
Thus, by definition of increase, it follows that $\Delta_Q(i) = \frac{p_r}{2}$.

3. Finally, assume that $i = i_l^{(r)}$. Since $i_l^{(r)} \in \mathcal{A}_{\overrightarrow{Q^{(r-1)}}} \setminus \mathcal{S} \mid_\alpha \left(Q^{(r-1)}, Q\right]$, by condition (A/1) shown above, we have that $r_Q(i_l^{(r)}) = b_{r-1} + \frac{p_{r-1}}{2} = b_r$. By construction of $G(\mathsf{Sched})$, $i_l^{(r)}$ traverses the following edges.
   - edge $e_l^{(r)}$ with capacity $c(e_l^{(r)}) = 2\,b_r + \frac{p_r}{2}$;
   - edge $e^{(r')} \in E^{(r')}(\mathsf{Sched})$ for each integer $r'$, $1 \le r' < r$, in case $r > 1$.

   We next calculate separately the increases for $i_l^{(r)}$ in $Q$ allowed by these edges.
   (a) First consider edge $e_l^{(r)}$, which is also traversed by session $i_f^{(r)}$. Since $i_l^{(r)}$ is scheduled in front of $\overrightarrow{Q}$, it follows that $i_f^{(r)} \in \mathcal{S} \mid_\alpha \left(Q^{(r-1)}, Q\right]$. Thus, condition (A/1) shown above implies that $r_Q(i_f^{(r)}) = b_r + \frac{p_r}{2}$. By Lemma 7.1,

$$\begin{aligned}\Delta_Q(i_l^{(r)}, e_l^{(r)}) &= c(e_l^{(r)}) - r_Q(i_l^{(r)}) - \min\left\{\frac{c(e_l^{(r)})}{2}, r_Q(i_f^{(r)})\right\}\\ &= 2\,b_r + \frac{p_r}{2} - b_r - \min\left\{b_r + \frac{p_r}{4}, b_r + \frac{p_r}{2}\right\}\\ &= \frac{p_r}{4}.\end{aligned}$$

   (b) Finally, consider any edge $e^{(r')}$ for some integer $r'$, $1 \le r' < r$, in case $r > 1$, which is also traversed by session $i_f^{(r')} \notin \{i_f^{(r)}, i_l^{(r)}\}$. Induction hypothesis of induction on $r$ (condition (B/3)) implies that $i_f^{(r')} \notin \mathcal{A}_{\overrightarrow{Q^{(r-1)}}}$, so that $i_f^{(r')} \notin \mathcal{A}_Q$. It follows that $r_{Q^{(r-1)}}(i_f^{(r')}) = r_Q(i_f^{(r')})$. Recall that $r_Q(i_l^{(r)}) = b_{r-1} + \frac{p_{r-1}}{2}$. On the other hand, by the induction hypothesis of induction on $r$ (condition (B/2)), we have that $r_{Q^{(r-1)}}(i_l^{(r)}) = b_{r-1} + \frac{p_{r-1}}{2}$. Hence, $r_Q(i_l^{(r)}) = r_{Q^{(r-1)}}(i_l^{(r)})$. It now follows that $\mathit{resid}_Q(e^{(r')}) = \mathit{resid}_{Q^{(r-1)}}(e^{(r')})$. By the induction hypothesis of induction on $r$ (condition (B/4/a)), $\mathit{resid}_{Q^{(r-1)}}(e^{(r')}) \ge \frac{p_{r-1}}{4}$, so that $\mathit{resid}_Q(e^{(r')}) \ge \frac{p_{r-1}}{4}$ as well. By Lemma 7.2, it follows that $\Delta_Q(i_l^{(r)}, e^{(r')}) \ge \frac{p_{r-1}}{4} = p_r$.

   Thus, by definition of increase, it follows that $\Delta_Q(i_l^{(r)}) = \frac{p_r}{4}$. □

The proof of part (A) is now complete. We continue with the proof of part (B).

*Proof of part* (B).

*Proof of* (B/1). Assume, by way of contradiction, that $l_r = \infty$. Then there exists some session $i \in \mathcal{A}_{\overrightarrow{Q^{(r-1)}}}$ which is not scheduled in front of any state following $Q^{(r-1)}$. By condition (A/5) shown above, it follows that for any state $Q$ such that $Q^{(r-1)} \overset{\alpha}{\longmapsto} Q$, $\Delta_Q(i) \ge \frac{p_r}{4} > 0$ (since $p_r > 0$). Thus, by the `update` operation, the rate of session $i$ can be increased without causing any decrease to the rate of any other session with smaller rate. This is in contradiction to max-min fairness. □

Condition (B/1) establishes that state $Q^{(r)}$ is well defined. Recall that in part (A) we considered all states from $Q^{(r-1)}$ through $\overleftarrow{Q^{(r)}}$. We now explore further properties of $Q^{(r)}$.

*Proof of* (B/2). Recall that session $i_l^{(r)}$ is scheduled in front of $Q^{(r)}$. By condition (A/1) shown above, $r_{\overleftarrow{Q^{(r)}}}(i_f^{(r)}) = b_r + \frac{p_r}{2}$, while $r_{\overleftarrow{Q^{(r)}}}(i_l^{(r)}) = b_{r-1} + \frac{p_{r-1}}{2} = b_r$. Moreover, by condition (A/5) shown above, $\Delta_{\overleftarrow{Q^{(r)}}}(i_l^{(r)}) = \frac{p_r}{4}$. Thus, by the `update` operation, $r_{Q^{(r)}}(i_l^{(r)}) = r_{\overleftarrow{Q^{(r)}}}(i_l^{(r)}) + \Delta_{\overleftarrow{Q^{(r)}}}(i_l^{(r)}) = b_r + \frac{p_r}{4}$; moreover, $r_{Q^{(r)}}(i_f^{(r)}) = \min\{r_{Q^{(r)}}(i_l^{(r)}), r_{\overleftarrow{Q^{(r)}}}(i_f^{(r)})\} = \min\left\{b_r + \frac{p_r}{4}, b_r + \frac{p_r}{2}\right\} = b_r + \frac{p_r}{4}$.

Consider now any session $i \in \mathcal{A}_{\overrightarrow{Q^{(r-1)}}} \setminus \{i_f^{(r)}, i_l^{(r)}\}$. By definition of $Q^{(r)}$, $i \in \mathcal{S}\,|_{\alpha}\,(Q^{(r-1)}, \overleftarrow{Q^{(r)}}]$. Thus, condition (A/1) shown above implies that $r_{\overleftarrow{Q^{(r)}}}(i) = b_r + \frac{p_r}{2}$. Since, by construction of $G(\mathsf{Sched})$, $i \cap i_l^{(r)} = \emptyset$, the `update` operation implies that $r_{Q^{(r)}}(i) = r_{\overleftarrow{Q^{(r)}}}(i) = b_r + \frac{p_r}{2}$. □

*Proof of* (B/3). By condition (A/4) shown above, it follows that fair shares of all active edges are preserved from $\overleftarrow{Q^{(r)}}$ to $Q^{(r)}$. Since $\mathsf{Alg}$ is bottleneck and the termination condition for bottleneck algorithms is a predicate on session rates and fair shares, it follows that the only sessions that are candidates to be terminated in state $Q^{(r)}$ are those whose rates are changed in $Q^{(r)}$; by conditions (A/1) and (B/1) shown above, these are sessions $i_f^{(r)}$ and $i_l^{(r)}$.

By condition (A/3/a) shown above, edge $e_l^{(r)}$ is a bottleneck edge for $\overleftarrow{Q^{(r)}}$. Condition (A/4) shown above implies that both $i_f^{(r)}, i_l^{(r)} \in \mathcal{A}_{Q^{(r)}}$, so that $e_l^{(r)} \in AE_{Q^{(r)}}$. Thus, Proposition 4.8 (condition (2)) implies that $e_l^{(r)}$ is a bottleneck edge for $Q^{(r)}$. Recall that $FS_{Q^{(r)}}(e_l^{(r)}) = FS_{\overleftarrow{Q^{(r)}}}(e_l^{(r)})$. Thus, by condition (A/2) shown above, it follows that $FS_{Q^{(r)}}(e_l^{(r)}) = b_r + \frac{p_r}{4}$. By condition (B/2) shown above, this implies that $r_{Q^{(r)}}(i_f^{(r)}) = r_{Q^{(r)}}(i_l^{(r)}) = FS_{Q^{(r)}}(e_l^{(r)})$. Since $\mathsf{Alg}$ is bottleneck, it follows that $i_f^{(r)}, i_l^{(r)} \in \mathsf{Term}\left(Q^{(r)}\right)$. □

*Proof of* (B/4). Take any session $i \in \mathcal{A}_{\overrightarrow{Q^{(r-1)}}} \setminus \{i_f^{(r)}, i_l^{(r)}\}$, and fix any integer $r'$, $1 \leq r' \leq r$. The proof is by case analysis on $r'$.

1. First consider the case where $r' = r$. By construction of $G(\mathsf{Sched})$, edge $e_i^{(r)}$ with capacity $c(e_i^{(r)}) = 2\,b_r + p_r$ is traversed by sessions $i_f^{(r)}$ and $i$. So,

$$\begin{aligned}
&\mathit{resid}_{Q^{(r)}}(e_i^{(r)})\\
&\quad = c(e_i^{(r)}) - r_{Q^{(r)}}(i_f^{(r)}) - r_{Q^{(r)}}(i)\\
&\quad = 2\,b_r + p_r - \left(b_r + \frac{p_r}{4}\right) - \left(b_r + \frac{p_r}{2}\right) \quad \text{(by condition (B/2) shown above)}\\
&\quad = \frac{p_r}{4},
\end{aligned}$$

which establishes condition (B/4/a) in this case.
We continue with condition (B/4/b). By condition (B/3) shown above, $i_f^{(r)} \notin$

$\mathcal{A}_{\overrightarrow{Q^{(r)}}}$, while $i \in \mathcal{A}_{\overrightarrow{Q^{(r)}}}$, so that $|\mathcal{A}_{\overrightarrow{Q^{(r)}}}|e_i^{(r)}| = 1$. Thus,

$$\begin{aligned}
&FS_{\overrightarrow{Q^{(r)}}}(e_i^{(r)})\\
&= \frac{c(e_i^{(r)}) - allot_{\overrightarrow{Q^{(r)}}}(e_i^{(r)})}{\left|\mathcal{A}_{\overrightarrow{Q^{(r-1)}}} \mid e_i^{(r)}\right|}\\
&= \frac{2b_r + p_r - (b_r + \frac{p_r}{4})}{1} \text{ (by conditions (B/1) and (B/3) shown above)}\\
&= b_r + \frac{3}{4}p_r,
\end{aligned}$$

which establishes condition (B/4/b) in this case.

2. Now consider the case where $r' < r$. Clearly, $r > 1$ in this case, so that the induction hypothesis of induction on $r$ is nonempty. Fix any edge $e_i^{(r')} \in E^{(r')}(\mathsf{Sched})$. Condition (B/3) shown above implies that $i_f^{(r')} \notin \mathcal{A}_{\overrightarrow{Q^{(r-1)}}}$, so that $r_{Q^{(r)}}(i_f^{(r')}) = r_{Q^{(r-1)}}(i_f^{(r')})$. By condition (B/2) shown above, $r_{Q^{(r)}}(i) = b_r + \frac{p_r}{2}$, while by the induction hypothesis of induction on $r$ (condition (B/2)), $r_{Q^{(r-1)}}(i) = b_{r-1} + \frac{p_{r-1}}{2} = b_r$. Thus,

$$\begin{aligned}
&resid_{Q^{(r-1)}}(e_i^{(r')}) - resid_{Q^{(r)}}(e_i^{(r')})\\
&= (c(e_i^{(r')}) - r_{Q^{(r-1)}}(i_f^{(r')}) - r_{Q^{(r-1)}}(i))\\
&\quad - (c(e_i^{(r')}) - r_{Q^{(r)}}(i_f^{(r')}) - r_{Q^{(r)}}(i))\\
&= (r_{Q^{(r)}}(i) - r_{Q^{(r-1)}}(i)) - (r_{Q^{(r)}}(i_f^{(r')}) - r_{Q^{(r-1)}}(i_f^{(r')}))\\
&= \left(b_r + \frac{p_r}{2}\right) - b_r - 0\\
&= \frac{p_r}{2}.
\end{aligned}$$

By the induction hypothesis of induction on $r$ (condition (B/4/a)), it follows that $resid_{Q^{(r-1)}}(e_i^{(r')}) \geq \frac{p_{r-1}}{4} = p_r$. Thus, $resid_{Q^{(r)}}(e_i^{(r')}) \geq p_r - \frac{p_r}{2} > \frac{p_r}{4}$, which establishes condition (B/4/a) in this case.

We continue with condition (B/4/b). Recall that $i_f^{(r')} \notin \mathcal{A}_{\overrightarrow{Q^{(r-1)}}}$, so that $i_f^{(r')} \notin \mathcal{A}_{\overrightarrow{Q^{(r)}}}$, and $r_{\overrightarrow{Q^{(r-1)}}}(i_f^{(r')}) = r_{\overrightarrow{Q^{(r)}}}(i_f^{(r')})$; on the other hand, $i \in \mathcal{A}_{\overrightarrow{Q^{(r)}}}$, so that $i \in \mathcal{A}_{\overrightarrow{Q^{(r-1)}}}$. It follows that $FS_{\overrightarrow{Q^{(r-1)}}}(e_i^{(r')}) = FS_{\overrightarrow{Q^{(r)}}}(e_i^{(r')})$. By the induction hypothesis of induction on $r$ (condition (B/4/b)), it follows that $FS_{\overrightarrow{Q^{(r-1)}}}(e_i^{(r')}) \geq b_{r-1} + \frac{3\,p_{r-1}}{4} = b_r + p_r$. Thus, $FS_{\overrightarrow{Q^{(r)}}}(e_i^{(r')}) \geq b_r + p_r > b_r + \frac{3\,p_r}{4}$, which establishes condition (B/4/b) in this case. □

The proof of Proposition 7.3 is now complete. □

We are now ready to prove the following.

THEOREM 7.4 (lower bound for oblivious algorithms). *Assume that* $\mathsf{Alg}$ *is optimistic, oblivious, and bottleneck, and that it computes the max-min fair rate vector. Then* $\mathcal{U}_{\mathsf{Alg}} \geq \frac{dn}{4} + \frac{n}{2}$.

*Proof.* We will show that $\mathcal{U}_{\mathsf{Alg}}(\langle G(\mathsf{Sched}), \mathcal{S}\rangle) = \frac{dn}{4} + \frac{n}{2}$, where $\mathsf{Alg} = \langle \mathsf{Sched}, \mathsf{Term}\rangle$ and $G(\mathsf{Sched})$ is the network constructed in section 6. To do so, we calculate

$\mathcal{U}_{\mathsf{Alg}}(\langle G\,(\mathsf{Sched})\,, \mathcal{S}_j\rangle)$ for any particular cluster $\mathcal{S}_j$, $j \geq 1$, and we add up over all $\frac{n}{d}$ clusters. By Proposition 7.3 (condition (B/1)), the execution of $\mathsf{Alg}$ on $G\,(\mathsf{Sched} \mid \mathcal{S}_j)$ is divided into $\frac{d}{2}$ execution epochs $\alpha^{(1)}, \ldots, \alpha^{(d/2)}$ such that all active sessions are scheduled at least once in each epoch; condition (B/3) implies that only two such sessions are terminated in each epoch. Thus, at least $d - 2\,(r-1)$ `update` operations are executed in $\alpha^{(r)}$, $1 \leq r \leq \frac{d}{2}$. Summing up over all clusters and execution epochs yields that $\mathcal{U}_{\mathsf{Alg}}\,(G, \mathcal{S}) \geq \sum_{j=1}^{n/d} \sum_{r=1}^{d/2} (d - 2(r-1)) = \frac{nd}{4} + \frac{n}{2}$, as needed. ∎

**7.2. Partially oblivious algorithms.** In this section, we present a lower bound of $\Omega(dn)$ on the convergence complexity of any optimistic, partially oblivious, and bottleneck algorithm that computes the max-min rate vector.

For the sake of clarity of presentation, we first consider the special case where $d = n$. (We remark that this is the case where the lower bound to be shown is maximum over all possible values of $d$, $1 \leq d \leq n$.) The backbone of our proof for this case will be a simulation lemma that establishes a correspondence between the executions of any partially oblivious algorithm and a suitable oblivious algorithm on the network constructed from the latter as in section 6 (assuming $d = n$). We will then use the simulation lemma to prove a lower bound of $\Omega(n^2)$ for this case. We finally consider the general case of arbitrary $d$; we state the $\Omega(dn)$ lower bound for it and discuss its proof, which relies on a generalization of the simulation lemma to general $d$. We start with the simulation lemma for the case $d = n$.

PROPOSITION 7.5 (simulation lemma). *Assume that* $\mathsf{Alg}$ *is optimistic, partially oblivious, and bottleneck, and that it computes the max-min fair rate vector. Then there exists some optimistic, oblivious, and bottleneck algorithm* $\mathsf{OAlg} = \langle \mathsf{OSched}, \mathsf{OTerm}\rangle$ *such that there exist execution prefixes* $\alpha = \alpha_1 \cdot \alpha_2 \cdot \ldots \cdot \alpha_{d/2}$ *of* $\mathsf{OAlg}$ *and* $\beta = \beta_1 \cdot \beta_2 \cdot \ldots \cdot \beta_{d/2}$ *of* $\mathsf{Alg}$*, on network* $G\,(\mathsf{OSched})$*, such that for each integer* $r$*,* $1 \leq r \leq \frac{d}{2}$*, the following conditions hold:*

(1) $\alpha_r$ *and* $\beta_r$ *are identical;*

(2) *all sessions active in* $\mathit{first}\,(\beta_r)$ *are scheduled at least once in* $\beta_r$ *and only two of them terminate in* $\beta_r$*.*

We first provide an informal, high-level outline of our proof. Given any partially oblivious algorithm, we derive some oblivious algorithm such that the executions of the two algorithms on the network constructed from the oblivious one in section 6 are identical. The execution of the oblivious algorithm is described by Proposition 7.3; since the two executions are shown to be identical, this will eventually imply the claimed lower bound for the partially oblivious algorithm. In order to derive the oblivious algorithm with the required properties, we start with the set of all $n$-epoch, oblivious algorithms; we inductively restrict this set until it contains only algorithms that are compatible with the partially oblivious algorithm (with respect to their schedulers) and perform sufficiently many `update` operations. All intermediate sets of oblivious algorithms contain only algorithms that induce executions identical to the partially oblivious algorithm up to each intermediate point. In more detail, the partial execution of each oblivious algorithm in any intermediate set on the network constructed from the algorithm in section 6 is identical to the partial execution of the partially oblivious algorithm on the same network. Each individual intermediate step is extended progressively by considering the session scheduled next by the partially oblivious algorithm and restricting the intermediate set of oblivious algorithms to those that schedule the same session next. (The construction follows an outer induction on execution epochs and an inner induction on states within each epoch.)

*Proof.* For each index $r$, $1 \leq r \leq \frac{d}{2}$, we will construct a set $\mathbf{A}^{(r)}$ of optimistic, oblivious, and bottleneck algorithms that compute the max-min fair rate vector such that for each algorithm $\mathsf{OAlg} \in \mathbf{A}^{(r)}$, there exist execution prefixes $\alpha(\mathsf{OAlg}) = \alpha_1(\mathsf{OAlg}) \dots \alpha_r(\mathsf{OAlg})$ of $\mathsf{OAlg}$ and $\beta(\mathsf{OAlg}) = \beta_1(\mathsf{OAlg}) \dots \beta_r(\mathsf{OAlg})$ of $\mathsf{Alg}$ on network $G(\mathsf{OSched})$ so that

(1) for each algorithm $\mathsf{OAlg} \in \mathbf{A}^{(r)}$ for each integer $r'$, $1 \leq r' \leq r$, $\alpha_{r'}(\mathsf{OAlg})$ and $\beta_{r'}(\mathsf{OAlg})$ are identical;
(2) $\alpha(\mathsf{OAlg})$ is identical over all algorithms $\mathsf{OAlg} \in \mathbf{A}^{(r)}$;
(3) $\beta(\mathsf{OAlg})$ is identical over all algorithms $\mathsf{OAlg} \in \mathbf{A}^{(r)}$;
(4) for each algorithm $\mathsf{OAlg} \in \mathbf{A}^{(r)}$ for each integer $r'$, $1 \leq r' \leq r$, all sessions active in state *first* $(\beta_{r'}(\mathsf{OAlg}))$ are scheduled at least once in $\beta_{r'}(\mathsf{OAlg})$, but only two of them terminate in $\beta_{r'}(\mathsf{OAlg})$.

Any algorithm $\mathsf{OAlg}$ in the final set $\mathbf{A}^{(d/2)}$ will satisfy the claim due to conditions (1) and (4) guaranteed by the construction.

The construction will be by induction on $r$, $1 \leq r \leq \frac{d}{2}$. The construction employs the set of all optimistic, oblivious, bottleneck, $n$-epoch algorithms, which we denote as $\mathbf{A}^{(0)}$. By Theorem 5.2, any algorithm from $\mathbf{A}^{(0)}$ computes the max-min fair rate vector. Our construction will progressively restrict the set $\mathbf{A}^{(0)}$ so that $\mathbf{A}^{(d/2)} \subseteq \cdots \subseteq \mathbf{A}^{(r)} \subseteq \cdots \subseteq \mathbf{A}^{(0)}$. Hence, for each integer $r$, $1 \leq r \leq \frac{d}{2}$, any algorithm from $\mathbf{A}^{(r)}$ computes the max-min fair rate vector. This property will be needed later, whenever we use Proposition 7.3, which assumes it. For each algorithm $\mathsf{OAlg} \in \mathbf{A}^{(0)}$, consider the initial state $Q_0(\mathsf{OAlg})$ of network $G(\mathsf{OSched})$ (with all rates zero and all sessions active). Set $\alpha_0(\mathsf{OAlg}) := Q_0(\mathsf{OAlg})$ and $\beta_0(\mathsf{OAlg}) := Q_0(\mathsf{OAlg})$. For the sake of shortening the construction, we merge the construction for the basis case (where $r = 1$) and the construction for the induction step. Thus, the case $r = 1$ will be treated separately (where needed) along the construction for the induction step.

Fix any integer $r$, $1 \leq r \leq \frac{d}{2}$, and assume inductively that we have constructed a set $\mathbf{A}^{(r-1)}$ with the required properties. Notice that if $r = 1$, then $\mathbf{A}^{(r-1)} = \mathbf{A}^{(0)}$ and the induction hypothesis is empty.

For the induction step, we construct a set $\mathbf{A}^{(r)} \subseteq \mathbf{A}^{(r-1)}$ with the required properties. This construction is progressive and uses induction on states, starting with *last* $(\beta(\mathsf{OAlg}))$ for any algorithm $\mathsf{OAlg} \in \mathbf{A}^{(r-1)}$. We will prove that conditions (1), (2), and (3) are preserved along the inductive construction on states, while (4) holds when the construction of $\mathbf{A}^{(r)}$ is complete.

For the basis case (of induction on states), take $\mathbf{A}^{(r)} := \mathbf{A}^{(r-1)}$; for any algorithm $\mathsf{OAlg} = \langle \mathsf{OSched}, \mathsf{OTerm} \rangle \in \mathbf{A}^{(r-1)}$, set $\alpha_r(\mathsf{OAlg}) = \mathit{last}(\alpha_{r-1}(\mathsf{OAlg}))$ and $\beta_r(\mathsf{OAlg}) = \mathit{last}(\beta_{r-1}(\mathsf{OAlg}))$. In case $r = 1$, by definition of initial state $(Q_0(\mathsf{OAlg}))$, $\alpha_1(\mathsf{OAlg})$ (resp., $\beta_1(\mathsf{OAlg})$) is identical for all algorithms $\mathsf{OAlg} \in \mathbf{A}^{(1)}$, implying (2) and (3). By construction, for each algorithm $\mathsf{OAlg} \in \mathbf{A}^{(1)}$, it trivially holds that $\alpha_1(\mathsf{OAlg})$ and $\beta_1(\mathsf{OAlg})$ are identical, which implies (1). In case $r > 1$, by the induction hypothesis of induction on $r$, conditions (1), (2), and (3) hold.

We now continue with the induction step (induction on states) of the inductive construction of $\mathbf{A}^{(r)}$. Take any algorithm $\mathsf{OAlg} \in \mathbf{A}^{(r)}$ and consider $\alpha(\mathsf{OAlg})$. (By the induction hypothesis of induction on states (condition (2)), the choice of $\mathsf{OAlg}$ does not matter.) In case $r = 1$, all sessions are active in state $\mathit{last}(\beta_{r-1}(\mathsf{OAlg})) = Q_0(\mathsf{OAlg})$. In case $r > 1$, since $r - 1 < \frac{d}{2}$, Proposition 7.3 (condition (B/3)) implies that the set of active sessions in state $\mathit{last}(\alpha_{r-1}(\mathsf{OAlg}))$ is nonempty; the induction hypothesis of induction on states (condition (1)) implies now that the set of active sessions in state $\mathit{last}(\beta_{r-1}(\mathsf{OAlg}))$ is nonempty as well. If all such sessions have been scheduled at least once in $\alpha_r(\mathsf{OAlg})$, then the inductive construction of $\mathbf{A}^{(r)}$ is complete and conditions (1), (2), and (3) hold by the induction hypothesis of induction on states. (Condition (4) will be shown later.) So assume that there exists at least one such active session $i\,(\mathsf{OAlg})$ that has not been scheduled in $\beta_r\,(\mathsf{OAlg})$. By the induction hypothesis of induction on states (condition (1)), $\alpha_r\,(\mathsf{OAlg})$ and $\beta_r\,(\mathsf{OAlg})$ are identical. It follows that $i\,(\mathsf{OAlg})$ has not been scheduled in $\alpha_r\,(\mathsf{OAlg})$ either. Proposition 7.3 (condition (A/5)) implies that the increase for session $i\,(\mathsf{OAlg})$ in state *last* $(\alpha_r\,(\mathsf{OAlg}))$ is nonzero. By the induction hypothesis of induction on states (condition (1)), it follows that the increase for session $i\,(\mathsf{OAlg})$ in state *last* $(\beta_r\,(\mathsf{OAlg}))$ is nonzero as well. By the `update` operation, it follows that an increase to the rate of $i\,(\mathsf{OAlg})$ in *last* $(\beta_r\,(\mathsf{OAlg}))$ is possible without decreasing the rate of any other session. So, max-min fairness has not been reached yet and some session rate must change. However, a session rate changes only when an active session is scheduled. Since $\mathsf{Alg}$ computes the max-min fair rate vector, it follows that at least one active session in *last* $(\beta_r\,(\mathsf{OAlg}))$ is scheduled after *last* $(\beta_r\,(\mathsf{OAlg}))$. An inductive application of this argument implies that all sessions active in *first* $(\beta_r\,(\mathsf{OAlg}))$ are scheduled at least once in $\beta_r\,(\mathsf{OAlg})$. It follows that the inductive construction of $\mathbf{A}^{(r)}$ eventually terminates.

Denote by $i'\,(\mathsf{OAlg})$ the session scheduled by $\mathsf{Alg}$ immediately after *last* $(\beta_r\,(\mathsf{OAlg}))$ on network $G\,(\mathsf{OSched})$. By the induction hypothesis of induction on states (condition (3)), it follows that *last* $(\beta_r\,(\mathsf{OAlg}))$ is identical for all $\mathsf{OAlg} \in \mathbf{A}^{(r)}$. Since $\mathsf{Alg}$ is partially oblivious, this implies that $i'\,(\mathsf{OAlg})$ is identical for all $\mathsf{OAlg} \in \mathbf{A}^{(r)}$ as well; so, denote it $i'$. Fix any algorithm $\mathsf{OAlg} \in \mathbf{A}^{(r)}$ and restrict $\mathbf{A}^{(r)}$ to the set of all optimistic, oblivious, bottleneck, $n$-epoch algorithms whose schedulers have prefix $\sigma\,(\alpha_r\,(\mathsf{OAlg}))\ i'$.

We now argue that $\mathbf{A}^{(r)}$ is nonempty and unique. Any sequence of sessions $\sigma\,(\beta_r\,(\mathsf{OAlg}))\,i'$ can be extended to an $n$-epoch scheduler (by appropriate padding). This implies that $\mathbf{A}^{(r)}$ is nonempty. By the induction hypothesis of induction on states (condition (3)), $\beta_r\,(\mathsf{OAlg})$ is identical over all algorithms $\mathsf{OAlg} \in \mathbf{A}^{(r)}$, so that $\sigma\,(\beta_r\,(\mathsf{OAlg}))\,i'$ is also identical over all $\mathsf{OAlg} \in \mathbf{A}^{(r)}$. It follows that the constructed $\mathbf{A}^{(r)}$ is unique.

Take any algorithm $\mathsf{OAlg} = \langle \mathsf{OSched}, \mathsf{OTerm} \rangle \in \mathbf{A}^{(r)}$.

- Define $Q\,(\mathsf{OAlg})$ to be the state that results when $\mathsf{Alg}$, starting from state *last* $(\beta_r\,(\mathsf{OAlg}))$ of the network $G\,(\mathsf{OSched})$, schedules session $i'$ on $G\,(\mathsf{OSched})$ and set $\beta_r\,(\mathsf{OAlg}) := \beta_r\,(\mathsf{OAlg}),\ i',\ Q\,(\mathsf{OAlg})$.
- Similarly, define $Q'\,(\mathsf{OAlg})$ to be the state that results when $\mathsf{OAlg}$, starting from *last* $(\alpha_r\,(\mathsf{OAlg}))$ of the network $G\,(\mathsf{OSched})$, schedules session $i'$ on $G\,(\mathsf{OSched})$ and set $\alpha_r\,(\mathsf{OAlg}) := \alpha_r\,(\mathsf{OAlg}),\ i',\ Q'\,(\mathsf{OAlg})$.

We now prove the required properties for the sets

$$\alpha\,(\mathsf{OAlg}) = \left\{ \alpha_1\,(\mathsf{OAlg}) \cdot \ldots \cdot \alpha_r\,(\mathsf{OAlg}) \ \mid\ \mathsf{OAlg} \in \mathbf{A}^{(r)} \right\}$$

and

$$\beta\,(\mathsf{OAlg}) = \left\{ \beta_1\,(\mathsf{OAlg}) \cdot \ldots \cdot \beta_r\,(\mathsf{OAlg}) \ \mid\ \mathsf{OAlg} \in \mathbf{A}^{(r)} \right\}$$

of execution prefixes.

We start by proving (1). Take any algorithm $\mathsf{OAlg} = \langle \mathsf{OSched}, \mathsf{OTerm} \rangle \in \mathbf{A}^{(r)}$. By the induction hypothesis of induction on states (condition (1)), $\mathsf{Alg}$ and $\mathsf{OAlg}$ start from identical states of the network $G\,(\mathsf{OSched})$; hence, scheduling the same session $(i')$ on the network $G\,(\mathsf{OSched})$ results in identical states, which inductively extends the claim.

We now treat (2) and (3). Since $i'(\mathsf{OAlg})$ is identical over all $\mathsf{OAlg} \in \mathbf{A}^{(r)}$, the induction hypothesis of induction on states (condition (2)) implies that the set of sessions scheduled in $\alpha_r(\mathsf{OAlg})$ is identical over all $\mathsf{OAlg} \in \mathbf{A}^{(r)}$. So, Proposition 7.3 (conditions (A/1) and (A/4)) implies that $Q'(\mathsf{OAlg})$ is identical over all $\mathsf{OAlg} \in \mathbf{A}^{(r)}$. Induction hypothesis of induction on states (condition (2)) and the fact that $i'(\mathsf{OAlg})$ is identical over all $\mathsf{OAlg} \in \mathbf{A}^{(1)}$ imply that $\alpha_r(\mathsf{OAlg})$ is identical over all $\mathsf{OAlg} \in \mathbf{A}^{(r)}$, which proves (2). Now (3) follows from (1) and (2).

We finally prove (4) for the set $\mathbf{A}^{(r)}$ constructed when the induction on states is complete. Fix any algorithm $\mathsf{OAlg} \in \mathbf{A}^{(r)}$. Recall that all sessions active in *first* $(\beta_r(\mathsf{OAlg}))$ are scheduled at least once in $\beta_r(\mathsf{OAlg})$; moreover, the termination condition used in the construction of $\mathbf{A}^{(r)}$ implies that $\beta_r(\mathsf{OAlg})$ is the shortest such fragment. By condition (1) shown above, this implies that $\alpha_r(\mathsf{OAlg})$ is the shortest execution fragment such that all sessions active in *first* $(\alpha_r(\mathsf{OAlg}))$ are scheduled at least once in $\alpha_r(\mathsf{OAlg})$. So, by Proposition 7.3 (condition (B/3)), only two sessions terminate in $\alpha_r(\mathsf{OAlg})$. Hence, the induction hypothesis of induction on states (condition (1)) implies that only two sessions terminate in $\beta_r(\mathsf{OAlg})$, as needed.

The proof of the simulation lemma is now complete. ☐

We are now ready to prove the lower bound for the case where $d = n$.

THEOREM 7.6 (lower bound for partially oblivious algorithms). *Assume that* $\mathsf{Alg}$ *is optimistic, partially oblivious, and bottleneck, and that it computes the max-min rate vector. Then* $\mathcal{U}_{\mathsf{Alg}} \geq \frac{n^2}{4} + \frac{n}{2}$.

*Proof.* Proposition 7.5 implies that there exists some optimistic, oblivious, and bottleneck algorithm $\mathsf{OAlg} = \langle \mathsf{OSched}, \mathsf{OTerm} \rangle$ such that there exists an execution prefix $\beta = \beta_1 \cdot \beta_2 \cdot \ldots \cdot \beta_{d/2}$ of $\mathsf{Alg}$ on network $G(\mathsf{OSched})$ such that for each integer $r$, $1 \leq r \leq \frac{d}{2}$, all sessions active in *first* $(\beta_r)$ are scheduled at least once in $\beta_r$ and only two of them terminate in $\beta_r$. Hence, at least $n - 2(r-1)$ `update` operations are executed in $\beta_r$, $1 \leq r \leq \frac{n}{2}$. Summing up over all $r$ epochs, $1 \leq r \leq \frac{n}{2}$, yields that $\mathcal{U}_{\mathsf{Alg}}(G(\mathsf{OSched}), \mathcal{S}) \geq \sum_{r=1}^{n/2}(n - 2(r-1)) = \frac{n^2}{4} + \frac{n}{2}$. ☐

As a direct generalization of the simulation lemma for the case $d = n$, we obtain the following.

PROPOSITION 7.7 (simulation lemma). *Assume that* $\mathsf{Alg}$ *is optimistic, partially oblivious, and bottleneck, and that it computes the max-min fair rate vector. Partition* $\mathcal{S}$ *into disjoint clusters* $\mathcal{S}_1, \ldots, \mathcal{S}_{n/d}$ *with* $d$ *sessions each. Then there exists some optimistic, oblivious, and bottleneck algorithm* $\mathsf{OAlg} = \langle \mathsf{OSched}, \mathsf{OTerm} \rangle$ *such that there exist execution prefixes* $\alpha$ *of* $\mathsf{OAlg}$ *and* $\beta$ *of* $\mathsf{Alg}$*, on network* $G(\mathsf{OSched})$*, such that for each cluster* $\mathcal{S}_j$*,* $1 \leq j \leq \frac{n}{d}$*,* $\alpha$ *and* $\beta$ *can be written as* $\alpha = \alpha_1 \cdot \alpha_2 \cdot \ldots \cdot \alpha_{\frac{d}{2}}$ *and* $\beta = \beta_1 \cdot \beta_2 \cdot \ldots \cdot \beta_{\frac{d}{2}}$ *so that for each integer* $r$*,* $1 \leq r \leq \frac{d}{2}$*, the following conditions hold:*

(1) $\alpha_r$ *and* $\beta_r$ *are identical;*

(2) *all sessions from* $\mathcal{S}_j$ *active in first* $(\beta_r)$ *are scheduled at least once in* $\beta_r$ *and only two of them terminate in* $\beta_r$.

The execution prefixes $\alpha$ and $\beta$ in Proposition 7.7 are constructed in a similar inductive manner to the corresponding prefixes in Proposition 7.5. The additional complications stem from the fact that the execution fragments $\alpha_r$ and $\beta_r$, $1 \leq r \leq \frac{d}{2}$, may now be different for each cluster; thus, the inductive construction in the proof of Proposition 7.5 needs to be adjusted in order to accommodate chopping off $\alpha$ and $\beta$ into the suitable execution fragments $\alpha_r$ and $\beta_r$, $1 \leq r \leq \frac{d}{2}$, for each particular cluster. We finally use Proposition 7.7 to show our final lower bound; its proof is similar to the one of Theorem 7.6 that used Proposition 7.5.

THEOREM 7.8 (lower bound for partially oblivious algorithms). *Assume that* Alg *is optimistic, partially oblivious, and bottleneck, and that it computes the max-min rate vector. Then* $\mathcal{U}_{\mathsf{Alg}} \geq \frac{dn}{4} + \frac{n}{2}$.

Consider the partially oblivious algorithm GlobalMinSched introduced by Afek, Mansour, and Ostfeld [1, section 4], whose scheduler chooses, for each state $Q$, the active session with the minimum rate. We note that [1, Theorem 4.3] implies an upper bound $\frac{|\mathcal{S}_j|\,(|\mathcal{S}_j|+1)}{2}$ on the number of `update` operations executed by GlobalMin on network $G$ with session set $\mathcal{S}_j$ for any particular cluster $\mathcal{S}_j$. So, $\mathcal{U}_{\mathsf{GlobalMin}} \leq \sum_{j\geq 1} \frac{|\mathcal{S}_j|\,(|\mathcal{S}_j|+1)}{2} \leq \frac{\max_{j\geq 1}|\mathcal{S}_j|+1}{2} \sum_{j\geq 1} |\mathcal{S}_j| = \frac{dn}{2} + \frac{n}{2}$. This implies that the lower bound established in Theorem 7.8 is tight (within a factor of 2).

## 8. Discussion and directions for further research.

We have presented a comprehensive collection of lower and upper bounds on the convergence complexity of optimistic, bottleneck, rate-based flow control algorithms, under varying degrees of the knowledge used by the scheduling component of the algorithms. In particular, we have defined and studied oblivious, partially oblivious, and nonoblivious algorithms. We have shown that, perhaps surprisingly, the classes of oblivious algorithms and partially oblivious algorithms collapse with respect to convergence complexity; we have also shown a convergence complexity separation between (partially) oblivious algorithms and nonoblivious algorithms. A more complete presentation of results for the model studied in this paper (and extensions of it) can be found in [8].

For the sake of completeness and comparison, we summarize in Table 1 all known lower and upper bounds on the convergence complexity of optimistic, bottleneck algorithms for rate-based flow control, established in this work and in the preceding work by Afek, Mansour, and Ostfeld [1]. We remark that RoundRobin represents an exponential improvement over the previous algorithm Arbitrary [1, section 6] for the class of oblivious algorithms we introduced. (The algorithm Arbitrary schedules sessions in any arbitrary way.)

TABLE 1
*Summary of known lower and upper bounds on convergence complexity for optimistic, bottleneck rate-based flow control algorithms.*

| Scheduler types | Lower bounds | Upper bounds |
|---|---|---|
| Oblivious | $\frac{dn}{4} + \frac{n}{2}$ | $\frac{dn}{2} + \frac{n}{2}$ (RoundRobin)<br>$\Theta(2^n)$ (Arbitrary [1]) |
| Partially oblivious | $\frac{dn}{4} + \frac{n}{2}$ | $\frac{dn}{2} + \frac{n}{2}$ (LocalMin or GlobalMin [1]) |
| Nonoblivious | $n$ | $n$ (Linear) |

Our work leaves open several important questions. The most obvious open question would be how to close the gap between the lower bound of $\frac{dn}{4} + \frac{n}{2}$ and the upper bound of $\frac{dn}{2} + \frac{n}{2}$ we have shown on the convergence complexity of oblivious algorithms.

The model considered in this work is simple and elegant, yet structured enough to capture several significant ingredients of distributed rate-based flow control; there remain, however, a number of significant practical issues untouched by our model and analysis. In the first place, we feel that the max-min fairness criterion may be undue in some realistic situations, where sessions have different demands. (Some results in this direction have been obtained in [10].) Second, the limitation to static sets of sessions is somehow restrictive; it would be significant to extend our model and techniques to handle set-up and take-down of sessions. Third, practical considerations may demand that rate-based flow control algorithms avoid too small or too large adjustments to session rates. Encompassing such practical considerations, and analyzing their impact on convergence complexity, into the framework of rate-based flow control algorithms is an interesting research problem.

Kleinberg, Rabani, and Tardos [20] formulated some natural approximations to max-min fairness and advocated them as suitable fairness conditions for certain routing and load balancing applications. It would be interesting to study the convergence complexity of such approximations within the framework of rate-based flow control algorithms.

**Acknowledgments.** Our thanks go to the anonymous *PODC* 1997 and *SIAM Journal on Computing* reviewers for their helpful comments.

## REFERENCES

[1] Y. Afek, Y. Mansour, and Z. Ostfeld, *Convergence complexity of optimistic rate based flow control algorithms*, J. Algorithms, 30 (1999), pp. 106–133.
[2] D. P. Bertsekas and R. G. Gallager, *Data Networks*, 2nd ed., Prentice-Hall, London, 1992.
[3] K. Bharath-Kumar and J. M. Jaffe, *A new approach to performance-oriented flow control*, IEEE Trans. Comm., 29 (1981), pp. 427–435.
[4] F. Bonomi and K. Fendick, *The rate-based flow control for available bit rate ATM service*, IEEE Network, 9 (1995), pp. 24–39.
[5] A. Charny, *An Algorithm for Rate Allocation in a Packet-Switching Network with Feedback*, Tech. Report MIT/LCS/TR-601, Laboratory for Computer Science, Massachusetts Institute of Technology, Cambridge, MA, 1994.
[6] A. Charny, D. D. Clark, and R. Jain, *Congestion control with explicit rate indication*, in Proceedings of the IEEE 30th International Conference on Communications, 1995, pp. 1954–1963.
[7] A. Charny, K. K. Ramakrishnan, and A. Lauck, *Time scale analysis and scalability issues for explicit rate allocation in ATM networks*, IEEE/ACM Trans. Networking, 4 (1996), pp. 569–581.
[8] P. Fatourou, *Algorithmic Foundations of Fair Rate-Based Flow Control*, Ph.D. thesis, Department of Computer Engineering and Informatics, University of Patras, Greece, 1999.
[9] P. Fatourou, M. Mavronicolas, and P. Spirakis, *The global efficiency of distributed, rate-based flow control algorithms*, in Proceedings of the 5th Colloquium on Structural Information and Communication Complexity, Carleton Scientific, Waterloo, ON, Canada, 1998, pp. 244–258.
[10] P. Fatourou, M. Mavronicolas, and P. Spirakis, *Max-min fair flow control sensitive to priorities*, in Proceedings of the 2nd International Conference on Principles of Distributed Systems, Hermes, 1998, pp. 45–59.
[11] E. Gafni and D. Bertsekas, *Dynamic control of session input rates in communication networks*, IEEE Trans. Automat. Control, 29 (1984), pp. 1009–1016.
[12] M. Gerla and L. Kleinrock, *Flow control: A comparative survey*, IEEE Trans. Comm., 28 (1980), pp. 553–574.
[13] E. Hahne, *Round-robin scheduling for maxmin fairness in data networks*, IEEE J. Selected Areas in Commun., 9 (1991), pp. 1024–1039.
[14] E. Hahne and R. G. Gallager, *Round-robin scheduling for fair flow control in data communication networks*, in Proceedings of the IEEE International Conference on Communications, 1986, pp. 103–107.
[15] H. Hayden, *Voice Flow Control in Integrated Packet Networks*, Tech. Report MIT/LIDS/TH/TR-601, Laboratory for Information and Decision Systems, Massachusetts Institute of Technology, Cambridge, MA, 1981.
[16] J. M. Jaffe, *Bottleneck flow control*, IEEE Trans. Comm., 29 (1981), pp. 954–962.
[17] J. M. Jaffe, *Flow control power is nondecentralizable*, IEEE Trans. Comm., 29 (1981), pp. 1301–1306.
[18] R. Jain, S. Kalyanaraman, and R. Viswanathan, *The OSU Scheme for Congestion Avoidance Using Explicit Rate Indication*, Tech. Report atm-forum/95-0883, Ohio State University, 1994.
[19] L. Kalampoukas, A. Varma, and K. K. Ramakrishnan, *An efficient rate allocation algorithm for ATM networks providing max-min fairness*, in Proceedings of the 6th IFIP Interna-

tional Conference on High Performance Networking, Chapman and Hall, London, 1995, pp. 143–154.

[20] J. Kleinberg, Y. Rabani, and É. Tardos, *Fairness in routing and load balancing*, in Proceedings of the 40th Annual IEEE Symposium on Foundations of Computer Science, 1999, pp. 568–578.

[21] H. Luss and D. R. Smith, *Resource allocation among competing entities: A lexicographic minimax approach*, Oper. Res. Lett., 5 (1986), pp. 227–231.

[22] J. Mosely, *Asynchronous Distributed Flow Control Algorithms*, Ph.D. thesis, Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, Cambridge, MA, 1984.

SIAM J. COMPUT.
Vol. 34, No. 5, pp. 1253–1270

# OPTIMAL TWO-STAGE ALGORITHMS FOR GROUP TESTING PROBLEMS[*]

ANNALISA DE BONIS[†], LESZEK GĄSIENIEC[‡], AND UGO VACCARO[†]

**Abstract.** Group testing refers to the situation in which one is given a set of objects $\mathcal{O}$, an unknown subset $\mathcal{P} \subseteq \mathcal{O}$, and the task of determining $\mathcal{P}$ by asking queries of the type "does $\mathcal{P}$ intersect $\mathcal{Q}$?", where $\mathcal{Q}$ is a subset of $\mathcal{O}$. Group testing is a basic search paradigm that occurs in a variety of situations such as quality control testing, searching in storage systems, multiple access communications, and data compression, among others. Group testing procedures have been recently applied in computational molecular biology, where they are used for screening libraries of clones with hybridization probes and sequencing by hybridization.

Motivated by particular features of group testing algorithms used in biological screening, we study the efficiency of two-stage group testing procedures. Our main result is the first optimal two-stage algorithm that uses a number of tests of the same order as the information-theoretic lower bound on the problem. We also provide efficient algorithms for the case in which there is a Bernoulli probability distribution on the possible sets $\mathcal{P}$, and an optimal algorithm for the case in which the outcome of tests may be unreliable because of the presence of "inhibitory" items in $\mathcal{O}$. Our results depend on a combinatorial structure introduced in this paper. We believe that it will prove useful in other contexts, too.

**Key words.** group testing, cover-free families

**AMS subject classifications.** 68Q25, 68P10, 68R05, 05D05

**DOI.** 10.1137/S0097539703428002

## 1. Introduction and contributions.

In group testing, the task is to determine the *positive* members of a set of objects $\mathcal{O}$ by asking subset queries of the form "does the subset $\mathcal{Q} \subseteq \mathcal{O}$ contain a positive object?" The answer to each query informs the tester whether or not the subset $\mathcal{Q}$ (in common parlance called a *pool*) has a nonempty intersection with the subset of positive members denoted by $\mathcal{P}$. A negative answer to this question informs the tester that all the items belonging to pool $\mathcal{Q}$ are negative, i.e., nonpositive. The aim of group testing is to identify the unknown subset $\mathcal{P}$ using as few queries as possible.

Group testing was originally introduced as a potential approach to economical mass blood testing [23]. However, due to its basic nature, it has been proven to be applicable in a surprising variety of situations, including quality control in product testing [52], searching files in storage systems [36], sequential screening of experimental variables [40], efficient contention resolution algorithms for multiple-access communication [36, 55], data compression [31], and computation in the data stream model [16]. Group testing has also exhibited strong relationships with several disciplines such as coding theory, information theory, complexity, computational geometry, and computational learning theory, among others.

[*]Received by the editors May 21, 2003; accepted for publication (in revised form) March 13, 2005; published electronically June 16, 2005. An extended abstract of this paper appeared in the *Proceedings of the* 30th *International Colloquium on Automata, Languages, and Programming*, Lecture Notes in Comput. Sci. 2719, J. C. M. Baeten et al., eds., Springer-Verlag, Berlin, 2003, pp. 81–96.

http://www.siam.org/journals/sicomp/34-5/42800.html

[†]Dipartimento di Informatica ed Applicazioni, Università di Salerno, 84081 Baronissi (SA), Italy (debonis@dia.unisa.it, uv@dia.unisa.it).

[‡]Department of Computer Science, The University of Liverpool, Liverpool, L69 7ZF, UK (leszek@csc.liv.ac.uk).

Probably the most important modern applications of group testing are in the realm of computational molecular biology, where it is used for screening libraries of clones with hybridization probes [5, 10, 9] and sequencing by hybridization [45, 49]. We refer to [6, 24, 29, 32] for an account of the fervent development of the area. The applications of group testing to biological screening present some distinctive features that pose new and challenging research problems. For instance, in the biological setting, screening one pool at the time is far more expensive than screening many pools in parallel. This strongly encourages the use of nonadaptive procedures for screening, that is, procedures in which all tests must be specified in advance without the tester knowing the outcomes of other tests. Instead, in adaptive group testing algorithms the tests are performed one by one, and the outcomes of previous tests are assumed known at the time of determining the current test. Unfortunately, nonadaptive group testing strategies are inherently much more costly than adaptive algorithms. This can be seen by observing that nonadaptive group testing algorithms are essentially equivalent to *superimposed codes* [25, 28, 36] (equivalently, cover-free families) and by using known nonexistential results on the latter [30, 25, 51]. More precisely, any nonadaptive group testing algorithm must use a number of tests $\Omega((p^2/\log p)\log n)$, where $p$ is the maximum number of positives and $n = |\mathcal{O}|$, and the best known algorithms use a number of tests $O(p^2 \log n)$. Closing the gap between the above upper and lower bounds would also imply solving a major open problem in extremal combinatorics, that is, that of estimating the exact maximum size of $p$-cover-free families [28]. In contrast, adaptive algorithms that use an optimal number of tests $O(p \log n)$ are known [24].

A nearly nonadaptive algorithm that is of considerable interest for screening problems is the so called *trivial two-stage algorithm* [37]. Such an algorithm proceeds in two stages. In the first stage certain pools are tested in parallel; in the second stage individual objects may be tested separately, depending on the outcomes of the first stage. The following quotation from [37, p. 371] well emphasizes the importance of such an algorithm:

> It is generally feasible to construct a number of pools (much fewer than the number of clones) initially by exploiting parallelism, but adaptive construction of pools with many clones during the testing procedure is discouraged. The technicians who implement the pooling strategies generally dislike even the 3-stage strategies that are often used. Thus the most commonly used strategies for pooling libraries of clones rely on a fixed but reasonably small set on nonsingleton pools. The pools are either tested all at once or in a small number of stages (usually at most 2) where the previous stage determines which pools to test in the next stage. The potential positives are then inferred and confirmed by testing of individual clones. In most biological applications each positive clone must be confirmed even if the pool results unambiguously indicate that it is positive. This is to improve the confidence in the results, given that in practice the tests are prone to errors.

Our first result is rather surprising: we prove that the best trivial two-stage algorithms are asymptotically as efficient as the best *fully adaptive* group testing algorithms, that is, algorithms with arbitrarily many stages. More precisely, we prove that there are trivial two-stage algorithms that determine all positives using a worst-case number of tests equal to the information-theoretic lower bound on the problem.

The information-theoretic lower bound is evidently a lower bound on the number of tests required by any algorithm, and it is independent from the number of performed stages.

There is another feature that differentiates biologically motivated group testing problems from traditional ones. In the classical scenario it is assumed that the presence of a single positive object in a pool is sufficient for the test to produce a positive result. However, recent work [29, 56] suggests that classical group testing procedures should take into account the possibility of the existence of "inhibitory items," that is, objects whose presence in the tested set could render the outcome of the test meaningless, as far as the detection of positive objects is concerned. In other words, if during the execution of an algorithm we tested a subset $\mathcal{Q} \subseteq \mathcal{O}$ containing positive items *and* inhibitory items, we would get the same answer as if $\mathcal{Q}$ did not contain any positive object. Similar issues were considered in [20], where further motivations for the problem were given. Our contribution to the latter issue is an algorithm that determines all positives in a set of objects, containing also up to a certain number of inhibitory items, and that uses the optimal worst-case number of tests, considerably improving the results of [21] and [29]. An interesting feature of our algorithm is that it can be implemented to run in only four stages.

We also consider the important situation in which a trivial two-stage strategy is used to find the set of positives, given that some prior information about them has been provided in terms of a Bernoulli probability distribution; that is, it is assumed that each object has a fixed probability $q$ of being positive. Usually $q$ is a function $q(n)$ of $n = |\mathcal{O}|$. This situation has received much attention [7, 8, 9, 43], starting from the important work [37]. The relevant parameter in this scenario is the average number of tests necessary to determine all positives. We prove that trivial two-stage strategies can asymptotically attain the information-theoretic lower bound for a large class of probability functions $q(n)$. It should be remarked that for two-stage group testing algorithms there are values of $q(n)$ for which lower bounds on the average number of tests exist that are better than the information-theoretic lower bounds [7, 37].

Our results depend on a combinatorial structure we introduce in this paper: $(k, m, n)$-selectors, to be formally defined in section 2. Our definition of $(k, m, n)$-selectors includes, as particular cases, well-known combinatorial objects such as superimposed codes [36, 28] and $k$-selectors [13]. Superimposed codes and $k$-selectors are very basic combinatorial structures and find application in a variety of areas such as cryptography, data security [39, 54], computational molecular biology [6, 21, 24, 32], multiaccess communication [24, 36], database theory [36], pattern matching [34], distributed coloring [41], circuit complexity [12], broadcasting in radio networks [13, 15], and other areas in computer science. We believe that our $(k, m, n)$-selectors will prove useful in several different areas.

**1.1. Previous results.** We refer the reader to the excellent monographs [1, 2, 24] for a survey of the vast literature on group testing. The papers [32, 37, 29] include a very nice account of the most important results on biologically motivated group testing problems. To the best of our knowledge, our paper is the first to address the problem of estimating the worst-case complexity of trivial two-stage group testing algorithms. The problem of estimating the minimum expected number of tests of trivial two-stage group testing algorithms when it is known that any item has a probability $p = p(n)$ of being positive has been studied in [7, 8, 9, 37, 43]. The papers most closely related to our results are [37, 8]. In particular, the paper [37] proves that, for several classes of probability functions $p(n)$, trivial two-stage group testing procedures are inherently more costly than fully adaptive group testing procedures (interestingly, we prove that this is not so in the worst-case analysis). The paper [8], with a real tour de force of the probabilistic method, provides a sharp estimate of the minimum expected number of tests of trivial two-stage procedures for an ample class of probability functions $p(n)$. Our approach is simpler and still allows us to obtain the correct order of magnitude of the minimum expected number of tests of the trivial two-stage group testing procedure for several classes of probability functions. A more detailed comparison of our results with those of [8] will be given at the end of section 4. Finally, the study of group testing in the presence of inhibitory items, the subject of section 5, was initiated in [29], continued in [21] and, under different models, also appears in [22] and [20].

**1.2. Summary of the results and structure of the paper.** In section 2 we formally define our main combinatorial tool, $(k, m, n)$-selectors, and we give bounds on their sizes. These bounds will be crucial for all our subsequent results. In section 3 we present a two-stage group testing algorithm with asymptotically optimal worst-case complexity. In section 3 we also present some related results of independent interest. For instance, we prove an $\Omega(k \log n)$ lower bound on the size of $k$-selectors defined in [13], improving on the lower bound $\Omega(\frac{k}{\log k} \log n)$ mentioned in [35]. This bound shows that the construction in [13] is optimal. Also in section 3 we establish an interesting link between our findings and the problem of learning Boolean functions in a constant number of rounds, in the sense of [17]. In section 4 we present our results on two-stage group testing algorithms for the case when a probability distribution on the possible set of positives is assumed. Finally, in section 5 we present a worst-case optimal algorithm for group testing in the presence of inhibitory items, improving on the algorithms given in [21, 29, 33]. We conclude the paper with a discussion of our main findings and of possible future lines of research.

## 2. $(k, m, n)$-selectors and bounds on their sizes.

In this section we introduce our main combinatorial tools, $(k, m, n)$-selectors. We point out their relationships with other well-known combinatorial objects and provide upper and lower bounds on their sizes.

DEFINITION 1. *Given integers $k$, $m$, and $n$, with $1 \leq m \leq k \leq n$, we say that a Boolean matrix $M$ with $t$ rows and $n$ columns is a $(k, m, n)$-selector if* any *submatrix of $M$ obtained by choosing $k$ out of $n$ arbitrary columns of $M$ contains at least $m$ distinct rows of the identity matrix $I_k$. The integer $t$ is the* size *of the $(k, m, n)$-selector.*

One can prove that $k$-cover-free families [28], disjunctive codes [24], superimposed codes [36], and strongly selective families [15, 13] correspond to our notion of a $(k + 1, k + 1, n)$-selector. The $k$-selectors of [13] coincide with our definition of $(2k, 3k/2 + 1, n)$-selectors.

We are interested in providing upper and lower bounds on the minimum size $t = t(k, m, n)$ of $(k, m, n)$-selectors. Upper bounds will be obtained by translating the problem into the hypergraph language. Given a finite set $X$ and a family $\mathcal{F}$ of subsets of $X$, a hypergraph is a pair $\mathcal{H} = (X, \mathcal{F})$. Elements of $X$ will be called vertices of $\mathcal{H}$, and elements of $\mathcal{F}$ will be called hyperedges of $\mathcal{H}$. A *cover* of $\mathcal{H}$ is a subset $T \subseteq X$ such that for any hyperedge $E \in \mathcal{F}$ we have $T \cap E \neq \emptyset$. The minimum size of a cover of $\mathcal{H}$ will be denoted by $\tau(\mathcal{H})$. A fundamental result by Lovász [42] implies that

$$(1) \qquad \tau(\mathcal{H}) < \frac{|X|}{\min_{E \in \mathcal{F}} |E|}(1 + \ln \Delta),$$

where $\Delta = \max_{x \in X} |\{E\colon\ E \in \mathcal{F} \text{ and } x \in E\}|$.

Essentially, Lovász proves that, by greedily choosing vertices in $X$ that intersect the maximum number of yet nonintersected hyperedges of $\mathcal{H}$, one obtains a cover of a size smaller than the right-hand side of (1). Our aim is to show that $(k, m, n)$-selectors are covers of properly defined hypergraphs. Lovász's result (1) will then provide us with the desired upper bound on the minimum selector size.

We shall proceed as follows. Let $X$ be the set of all binary vectors $\mathbf{x} = (x_1, \dots, x_n)$ of length $n$ containing $n/k$ 1's (the value $n/k$ is a consequence of an optimized choice whose justification can be skipped here). For any integer $i$, $1 \le i \le k$, let us denote by $\mathbf{a}_i$ the binary vector of length $k$ having all components equal to zero with the exception of the component in position $i$, that is, $\mathbf{a}_1 = (1, 0, \dots, 0)$, $\mathbf{a}_2 = (0, 1, \dots, 0)$,$\dots$, $\mathbf{a}_k = (0, 0, \dots, 1)$. Moreover, for any set of indices $S = \{i_1, \dots, i_k\}$, with $1 \le i_1 \le i_2 < \cdots < i_k \le n$, and for any binary vector $\mathbf{a} = (a_1, \dots, a_k) \in \{\mathbf{a}_1, \dots, \mathbf{a}_k\}$, let us define the set of binary vectors $E_{\mathbf{a},S} = \{\mathbf{x} = (x_1, \dots, x_n) \in X : x_{i_1} = a_1, \dots, x_{i_k} = a_k\}$. For any set $A \subseteq \{\mathbf{a}_1, \dots, \mathbf{a}_k\}$ of size $r$, $r = 1, \dots, k$, and any set $S \subseteq \{1, \dots, n\}$, with $|S| = k$, let us define $E_{A,S} = \bigcup_{\mathbf{a} \in A} E_{\mathbf{a},S}$. For any $r = 1, \dots, k$ we define $\mathcal{F}_r = \{E_{A,S} : A \subset \{\mathbf{a}_1, \dots, \mathbf{a}_k\}, |A| = r, \text{ and } S \subseteq \{1, \dots, n\}, |S| = k\}$ and the hypergraph $\mathcal{H}_r = (X, \mathcal{F}_r)$. We claim that *any* cover $T$ of $\mathcal{H}_{k-m+1}$ is a $(k, m, n)$-selector; that is, any submatrix of $k$ arbitrary columns of $T$ contains at least $m$ distinct rows of the identity matrix $I_k$. The proof is done by contradiction. Assume that there exists a set of indices $S = \{i_1, \dots, i_k\}$ such that the submatrix of $T$ obtained by considering only the columns of $T$ with indices $i_1, \dots, i_k$ contains *at most* $m - 1$ distinct rows of $I_k$. Let such rows be $\mathbf{a}_{j_1}, \dots, \mathbf{a}_{j_s}$, with $s \le m-1$; let $A$ be *any* subset of $\{\mathbf{a}_1, \dots, \mathbf{a}_k\} \setminus \{\mathbf{a}_{j_1}, \dots, \mathbf{a}_{j_s}\}$ of cardinality $|A| = k - m + 1$; and let $E_{A,S}$ be the corresponding hyperedge of $\mathcal{H}_{k-m+1}$. By construction we have that $T \cap E_{A,S} = \emptyset$, contradicting the fact that $T$ is a cover for $\mathcal{H}_{k-m+1}$.

The above proof that $(k, m, n)$-selectors coincide with the covers of $\mathcal{H}_{k-m+1}$ allows us to use Lovász's result (1) to give upper bounds on the minimum size of selectors.

THEOREM 1. *For any integers $k$, $m$, and $n$, with $1 \le m \le k < n$, there exists a $(k, m, n)$-selector of size $t$, with*

$$t < \frac{ek^2}{k - m + 1} \ln \frac{n}{k} + \frac{ek(2k - 1)}{k - m + 1},$$

*where $e = 2.7182\dots$ is the base of the natural logarithm.*

*Proof.* From the arguments preceding the theorem and (1), in order to upper bound $t$ we need only evaluate the quantities

$$|X|, \quad \min\{|E| : E \in \mathcal{F}_{k-m+1}\} \quad \text{and} \quad \Delta$$

for the hypergraph $\mathcal{H}_{k-m+1}$. We shall do it here.

By definition $X = \binom{n}{n/k}$. Moreover, each hyperedge $E_{A,S}$ of $\mathcal{H}_{k-m+1}$ is the union of $k - m + 1$ disjoint sets $E_{\mathbf{a},S}$; therefore it has cardinality

$$|E_{A,S}| = (k - m + 1) \cdot |E_{\mathbf{a},S}| = (k - m + 1) \binom{n - k}{n/k - 1}.$$

To compute $\Delta$, observe that each $\mathbf{x} \in X$ belongs to $\binom{n/k}{1}\binom{n-n/k}{k-1}$ distinct sets $E_{\mathbf{a},S}$, and each $E_{\mathbf{a},S}$ belongs to $\binom{k-1}{k-m}$ distinct hyperedges $E_{A,S}$. Therefore, for $\mathcal{H}_{k-m+1}$ we have

$$\Delta = \binom{k - 1}{k - m}\binom{n/k}{1}\binom{n - n/k}{k - 1}.$$

Hence one has

$$t < \frac{\binom{n}{n/k}}{(k-m+1)\binom{n-k}{n/k-1}}\left[1+\ln\binom{k-1}{k-m}\binom{n/k}{1}\binom{n-n/k}{k-1}\right]. \tag{2}$$

For $k \in \{1,2\}$, it is $\frac{\binom{n}{n/k}}{\binom{n-k}{n/k-1}} < 2k$, whereas for $k \geq 3$ it is

$$\begin{aligned}
\frac{\binom{n}{n/k}}{\binom{n-k}{n/k-1}} &= k\frac{n-1}{n-n/k}\cdot\frac{n-2}{n-n/k-1}\times\cdots\times\frac{n-k+1}{n-k-n/k+2}\\
&\leq k\left(\frac{n-k+1}{n-k-n/k+2}\right)^{k-1}\\
&= k\left(\frac{k(n-k+1)}{k(n-k+1)-(n-k)}\right)^{k-1}\\
&= k\left(1+\frac{n-k}{k(n-k+1)-(n-k)}\right)^{k-1}\\
&\leq k\left(1+\frac{1}{k-1}\right)^{k-1}\\
&< ek.
\end{aligned}$$

Moreover, using the well-known inequality $\binom{a}{b} \leq (ea/b)^b$, one can conclude

$$\begin{aligned}
\binom{k-1}{k-m}\binom{n/k}{1}\binom{n-n/k}{k-1} &\leq \left(\frac{k-1}{k-m}\right)^{k-m} e^{k-m}\frac{n}{k}\left(\frac{n-n/k}{k-1}\right)^{k-1} e^{k-1}\\
&= e^{2k-m-1}\left(1+\frac{m-1}{k-m}\right)^{k-m}\left(\frac{n}{k}\right)^k\\
&\leq e^{2k-m-1}\left(1+\frac{m}{k-m}\right)^{k-m}\left(\frac{n}{k}\right)^k\\
&\leq e^{2k-m-1}e^m\left(\frac{n}{k}\right)^k.
\end{aligned}$$

The theorem now follows from (2) and the above inequalities. □

*Remark.* Applying the above theorem to $(k,k,n)$-selectors, that is, to $(k-1)$-cover-free-families, one recovers the usual upper bound of $O(k^2\log n)$ on their sizes [25, 28]. Applying the above theorem to $(2k, 3k/2+1, n)$-selectors (that is, to $k$-selectors in the sense of [13]), one gets the same upper bound of $O(k\log n)$ on their sizes, with a better constant (22 versus 87). By concatenating $(k,\alpha k,n)$-selectors, $\alpha < 1$, of suitably chosen parameter $k$, one gets in a simple way the important combinatorial structure of [38], with the same asymptotic upper bound given therein, but our constants are much better (44 versus $\sim 5\cdot 10^5$, according to [11]).

To present our first lower bound on the size of $(k,m,n)$-selectors, we need to recall the definition of $(p,q)$-superimposed codes [21, 25].

Definition 2. *Given integers $p,q$, and $n$, with $p+q\leq n$, we say that a Boolean matrix $M$ with $n$ columns and $t$ rows is a $(p,q)$-superimposed code if for any choice of two subsets $P$ and $Q$ of columns of $M$, where $P\cap Q=\emptyset$, $|P|=p$, and $|Q|=q$, there exists a row in $M$ in which entries corresponding to the columns in $P$ contain at least one nonzero value and all entries corresponding to the columns in $Q$ are set* *to zero. The integers $n$ and $t$ are the size and the length of the $(p, q)$-superimposed code, respectively. The minimum length of a $(p, q)$-superimposed code of size $n$ will be denoted by $t_s(p, q, n)$.*

It can be seen that $(k, m, n)$-selectors are $(k - m + 1, m - 1)$-superimposed codes with additional properties. Therefore, lower bounds on the length of $(p, q)$-superimposed codes translates into lower bounds on selectors. The following theorem provides a lower bound on the length of $(p, q)$-superimposed codes, and its proof uses the techniques developed in [3, 51] to lower bound the length of classical $(1, q)$-superimposed codes. The theorem improves the results of [25]. A similar result also has been obtained by Dyachkov [27].

THEOREM 2. *For any positive integers $p$, $q$, and $n$, with $n > q^2/(4p)$ the minimum length $t_s(p, q, n)$ of a $(p, q)$-superimposed code of size $n$ is bounded from below by*

$$t_s(p,q,n) > \begin{cases} q \log\left(\frac{n}{e(p+q-1)}\right) & \text{if } 1 \le q < 2p, \\ \frac{p(\lfloor q/(2p)\rfloor)^2}{\log(eq^2/(4p))} \log\left(\frac{4(n-2(p-1)-q/2)}{eq^2}\right) & \text{if } q \ge 2p. \end{cases}$$

*Proof.* For $q < 2p$ the stated bound immediately follows from Proposition 2 of [25], which implies $t_s(p, q, n) \ge \lceil \log \binom{n}{q} - \log \binom{p+q-1}{q} \rceil$, and from the well-known inequality

$$\binom{m}{k} \le 2^{k \log(em/k)}. \tag{3}$$

Let us consider the case when $q \ge 2p$ and assume for the moment that $q$ is a multiple of $2p$.

Let $\mathcal{F}$ be the family associated with a $(p, q)$-superimposed code $M$ of length $t$ and size $n$; that is, $\mathcal{F}$ is a family of subset of $\{1, \ldots, t\}$, $|\mathcal{F}| = n$, such that the column vectors of $M$ are the characteristic vectors of the subsets in $\mathcal{F}$. By Definition 2 we have that for any $p+q$ pairwise different members $F_1, \ldots, F_p, G_1, \ldots, G_q$ it holds that

$$F_1 \cup \cdots \cup F_p \not\subseteq G_1 \cup \cdots \cup G_q. \tag{4}$$

To prove our lower bound we first transform $\mathcal{F}$ into a family $\mathcal{F}'$ with members of size at most $\lfloor 2t/q \rfloor$. As long as $\mathcal{F}$ contains a set $H$ of size larger than $\lfloor 2t/q \rfloor$, we remove $H$ from $\mathcal{F}$ and replace any other set $G \in \mathcal{F}$ with $G' = G \setminus H$. Since the elements of $\mathcal{F}$ are subsets of $\{1, \ldots, t\}$, this process terminates after at most $\ell \le q/2$ steps. If for some $G \in \mathcal{F}$ it results in $G' = G \setminus (H_1 \cup \cdots \cup H_\ell) = \emptyset$, then we remove $G'$ from $\mathcal{F}$. Notice that there are no more than $p - 1$ such sets; otherwise there would be $p$ sets whose union is contained in $H_1 \cup \cdots \cup H_\ell$, thus violating (4). The resulting family $\mathcal{F}'$ therefore has size $|\mathcal{F}'| \ge |\mathcal{F}| - p + 1 - q/2$ and it results in $|F'| \le \lfloor 2t/q \rfloor$ for any $F' \in \mathcal{F}'$. It is possible to see that

$$F'_1 \cup \cdots \cup F'_p \not\subseteq G'_1 \cup \cdots \cup G'_{q/2} \tag{5}$$

for any choice of $p + q/2$ sets $F'_1, \ldots, F'_p, G'_1, \ldots, G'_{q/2} \in \mathcal{F}'$. Assume by contradiction that $\mathcal{F}'$ contains $F'_1, \ldots, F'_p, G'_1, \ldots, G'_{q/2}$ such that $F'_1 \cup \cdots \cup F'_p \subseteq G'_1 \cup \cdots \cup G'_{q/2}$; then the union of the $p$ sets of $\mathcal{F}$ from which $F'_1, \ldots, F'_p$ were obtained would be contained in the union of at most $q$ sets given by $G'_1, \ldots, G'_{q/2}$ and the $\ell$ removed sets, contradicting (4).

For each subfamily $\{F'_1,\ldots,F'_p\} \subset \mathcal{F}'$, there is a set $X_{\{F'_1,\ldots,F'_p\}}$ with $|X_{\{F'_1,\ldots,F'_p\}}| \leq \lceil 4tp/q^2 \rceil$ such that $X_{\{F'_1,\ldots,F'_p\}} \subseteq F'_j$ for some $F'_j \in \{F'_1,\ldots,F'_p\}$, and $X_{\{F'_1,\ldots,F'_p\}} \not\subseteq F'$ for any $F' \in \mathcal{F}' \setminus \{F'_1,\ldots,F'_p\}$. Suppose by contradiction that there exists no such set $X_{\{F'_1,\ldots,F'_p\}}$; then we can partition each $F'_1,\ldots,F'_p$ into $\frac{q}{2p}$ subsets of size at most $\lceil 4tp/q^2 \rceil$ such that each of those subsets is contained in some member of $\mathcal{F}' \setminus \{F'_1,\ldots,F'_p\}$. It follows that each $F'_1,\ldots,F'_p$ is contained in the union of at most $\frac{q}{2p}$ other members of $\mathcal{F}' \setminus \{F'_1,\ldots,F'_p\}$, and consequently the union $F'_1 \cup \cdots \cup F'_p$ is contained in the union of at most $q/2$ other members of $\mathcal{F}'$, which is a contradiction to (5). Let us now group all but at most $p-1$ members of $\mathcal{F}'$ into $\lfloor |\mathcal{F}'|/p \rfloor$ pairwise disjoint subfamilies of size $p$ and let $\mathcal{P}$ denote the set of those subfamilies. From the above argument, the family $\mathcal{G} = \{X_{\{F'_1,\ldots,F'_p\}} : \{F'_1,\ldots,F'_p\} \in \mathcal{P}\}$ is a Sperner family (i.e., an antichain) consisting of $|\mathcal{P}|$ sets of size at most $\lceil 4tp/q^2 \rceil$, and consequently it is $|\mathcal{P}| = |\mathcal{G}| \leq \binom{t'}{\lceil 4tp/q^2 \rceil}$, where $t' = \left|\bigcup_{F' \in \mathcal{F}'} F'\right| \leq t$. It follows that $|\mathcal{F}'| \leq p|\mathcal{P}| + p - 1 \leq p\binom{t}{\lceil 4tp/q^2 \rceil} + p - 1$, and consequently,

$$|\mathcal{F}| \leq |\mathcal{F}'| + p - 1 + q/2 \leq p\binom{t}{\lceil 4tp/q^2 \rceil} + 2(p-1) + q/2.$$

From the above inequality and from inequality (3) it follows

$$|\mathcal{F}| \leq p 2^{\lceil 4pt/q^2 \rceil \log(eq^2/(4p))} + 2(p-1) + q/2. \tag{6}$$

Inequality (6) implies

$$t > \frac{q^2}{4p \log(eq^2/(4p))} \log\left(\frac{4(n - 2(p-1) - q/2)}{eq^2}\right). \tag{7}$$

To obtain the lower bound in the theorem, we need to deal with the case when $q \geq 2p$ is not a multiple of $2p$. In this case, we observe that any $(p,q)$-superimposed code is a $(p, 2p\lfloor q/2p \rfloor)$-superimposed code and, exploiting lower bound (7), we get the following lower bound that holds for any $q \geq 2p$ and $n \geq q^2/4p$:

$$t > \frac{(2p\lfloor \frac{q}{2p} \rfloor)^2}{4p \log(ep(\lfloor \frac{q}{2p} \rfloor)^2)} \log\left(\frac{4(n - 2(p-1) - p\lfloor \frac{q}{2p} \rfloor)}{e(2p\lfloor \frac{q}{2p} \rfloor)^2}\right). \tag{8}$$

Inequalities (6) and (8) imply the stated lower bound on $t_s(p,q,n)$. ☐

By setting $p = k - m + 1$ and $q = m - 1$ in the above lower bound, one obtains the following lower bound on the size of $(k,m,n)$-selectors.

Corollary 1. *For any integers $k$, $m$, and $n$, with $1 \leq m \leq k \leq n$, $n > \frac{(m-1)^2}{4(k-m+1)}$, the minimum size $t(k,m,n)$ of a $(k,m,n)$-selector is at least*

$$t(k,m,n) \geq \frac{(m-1)^2}{16(k-m+1)\log(\frac{(m-1)^2+(k-m+1)^2}{k-m+1})} \log \frac{4(n-m+1)}{e((m-1)^2+k-m+1)}. \tag{9}$$

## 3. Application of $(k,m,n)$-selectors to optimal two-stage group testing.

We have a set of objects $\mathcal{O}$, $|\mathcal{O}| = n$ and a subset $\mathcal{P} \subseteq \mathcal{O}$ of positives $|\mathcal{P}| \leq p$. Our task is to determine the members of $\mathcal{P}$ by asking subset queries of the form "does the subset $\mathcal{Q} \subseteq \mathcal{O}$ contain a positive object?" We focus on the so-called trivial two-stage algorithms. Recall that these algorithms consist of two stages: in the first stage certain pools are tested in parallel and in the second stage only individual objects are tested (always in parallel). Which individual objects are tested may depend on the outcomes of the first stage.

In the following we provide a two-stage algorithm that uses an asymptotically optimal number of tests. We associate each item of the input set $\mathcal{O}$ with a distinct column of a $(k, p{+}1, n)$-selector $M = [M(i, j)]$. Let $t$ denote the size of the $(k, p{+}1, n)$-selector. For $i = 1, \dots, t$, we define $T_i = \{j \in \{1, \dots, n\} : M(i, j) = 1\}$. The first stage of the algorithm consists of testing the $t$ pools $T_1, \dots, T_t$ in parallel. Let $\mathbf{f}$ denote the binary vector collecting the answers of the $t$ tests (here a "yes" answer to test $T_i$ corresponds to a 1 entry in the $i$th position of $\mathbf{f}$, and a "no" answer corresponds to a 0 entry). If $\mathcal{O}$ contains $q \le p$ positive items, then $\mathbf{f}$ is the Boolean sum of the $q$ columns associated with the $q$ positives. It is easy to see that, in addition to the columns associated with the positives items, there are at most $k - q - 1$ columns "covered" by $\mathbf{f}$, that is, have the 1's in a subset of the positions in which the vector $\mathbf{f}$ also has 1's. Let $y_1, \dots, y_q$ denote the $q$ positives. Assume by contradiction that there are more than $k - q - 1$ columns, other than those associated with $y_1, \dots, y_q$, which are covered by $\mathbf{f}$. Let $z_1, \dots, z_{k-q}$ denote $k - q \ge k - p$ such columns and let us consider the submatrix of $M$ consisting of $y_1, \dots, y_q, z_1, \dots, z_{k-q}$. By Definition 1, this submatrix contains at least $p + 1$ rows of the identity matrix $I_k$. At least one of these $p + 1$ rows of $I_k$ has a 1 in one of the columns $z_1, \dots, z_{k-q}$. Let $\ell$ denote the index of such a row. Since the columns associated with $y_1, \dots, y_q$ have the $\ell$th entry equal to 0, then the $\ell$th entry of $\mathbf{f}$ is 0, thus contradicting the hypothesis that $\mathbf{f}$ covers all columns $z_1, \dots, z_{k-q}$. Using this argument, one concludes that if we discard all columns not covered by $\mathbf{f}$, then we are left with $k - 1$ columns, $q$ of which correspond to the $q$ positives. Stage 2 consists of individually probing these $k - 1$ elements. The following theorem holds.

THEOREM 3. *Let $t$ be the size of a $(k, p+1, n)$-selector. There exists a two-stage group testing algorithm for finding up to $p$ positives out of $n$ items and that uses a number of tests equal to $t + k - 1$.*

From Theorems 1 and 3 we get the following.

COROLLARY 2. *For any integers $k$, $p$, and $n$, with $1 \le p < k \le n$, there exists a two-stage group testing algorithm for finding up to $p$ positives using a number of tests less than*

$$\frac{ek^2}{k-p} \ln \frac{n}{k} + \frac{ek(2k-1)}{k-p} + k - 1. \tag{10}$$

By optimizing the choice of $k$ to $k = 2p$ in (10), we get the main result of this section.

COROLLARY 3. *For any integers $p$ and $n$, with $1 \le p \le n$, there exists a two-stage group testing algorithm for finding up to $p$ positives using a number of tests less than*

$$4ep \ln \frac{n}{2p} + p(8e + 2) - 2e - 1 < 7.54p \log_2 \frac{n}{p} + 16.21p - 2e - 1.$$

The two-stage algorithm of the above corollary is asymptotically optimal because of the information-theoretic lower bound on the number of tests given by

$$\log_2 \binom{n}{p} > p \log_2 \frac{n}{p}, \tag{11}$$

that holds also for fully adaptive group testing algorithms.

Corollaries 2 and 3 also can be used to solve the open problem, mentioned in [29], of providing estimates for the maximum size of a search space in which it is possible to successfully search for $p$ positives, using at most $v$ pools and no more than $h$ tests involving single elements. This problem is equivalent to fixing the cardinality of the search space and trying to minimize the number of constructed pools. This equivalent problem is solved by the above corollaries since in our algorithm the number of pools coincides with the number of performed tests.

**3.1. Deriving a lower bound on the size of $(k, m, n)$-selectors via two-stage group testing.** Let $g(p, n)$ denote the minimum number of tests needed to identify $p$ positive items out of $n$ items by a group testing strategy. Theorem 3 and the information-theoretic lower bound (11) give

$$\log_2 \binom{n}{p} \le g(n, p) \le t(k, p+1, n) + k - 1,$$

from which we get the following result that also provides a lower bound on the size of $(k, m, n)$-selectors for values of $k$ and $m$ not covered by (9).

THEOREM 4. *For any integers $k$, $m$, and $n$, with $1 \le m \le k < n$, the minimum size $t(k, m, n)$ of a $(k, m, n)$-selector satisfies*

$$t(k, m, n) \ge \log \binom{n}{m-1} - k + 1 \ge (m-1) \log \frac{n}{m-1} - k + 1.$$

The bound given in Theorem 4 improves on the bound given in Corollary 1 for all values of $k$ and $m$ such that $m = \alpha k$ for constant $\alpha$, $0 < \alpha < 1$. Theorem 4 also implies a lower bound of $\Omega(k \log \frac{n}{k})$ on the size of the $k$-selectors of [13] (that is, of our $(2k, 3k/2 + 1, n)$-selectors), improving on the lower bound of $\Omega(\frac{k}{\log k} \log \frac{n}{k})$ mentioned in [35]. Our lower bound is optimal since it matches the upper bound on the size of $k$-selectors given in [13].

**3.2. A remark on learning monotone Boolean functions.** We consider here the well-known problem of exact learning of an unknown Boolean function of $n$ variables by means of membership queries, provided that at most $k$ of the variables (attributes) are relevant. This is known as attribute-efficient learning. By *membership queries* we mean the following [4]: The learner chooses a 0-1 assignment $x$ of the $n$ variables and gets the value $f(x)$ of the function at $x$. The goal is to learn (identify) the unknown function $f$ exactly, using a small number of queries. Typically, one assumes that the learner knows in advance that $f$ belongs to a restricted class of Boolean functions, since the exact learning problem in the full generality admits only trivial solutions. In this scenario, the group testing problem is equivalent to the problem of exactly learning an unknown function $f$, where it is known that $f$ is an $OR$ of at most $p$ variables.

Recently, in a series of papers [17, 18, 19] Damaschke studied the power of adaptive versus nonadaptive attribute-efficient learning. In this framework he proved that adaptive learning algorithms are more powerful than nonadaptive ones. More precisely, he proved that in general it is impossible to learn monotone Boolean functions with $k$ relevant variables in less than $\Omega(k)$ stages, if one insists that the total number of queries be of the same order as that used by the best fully adaptive algorithm (i.e., an algorithm that may use an arbitrary number of stages; see [17, 18] for details). In view of Damaschke's results, we believe it worthwhile to state our Corollary 3 in the following form.

COROLLARY 4. *Boolean functions made by the disjunction of at most $p$ variables are exactly learnable in only* two stages *by using a number of queries of the same order as that of the best fully adaptive learning algorithm.*

The above remark raises the interesting problem of how to characterize monotone Boolean functions which are "optimally" learnable in a constant number of stages. Another example of a class of functions optimally learnable in a constant number of stages will be given at the end of section 5.

## 4. Two-stage algorithms for probabilistic group testing.

In this section we assume that objects in $\mathcal{O}$, $|\mathcal{O}| = n$, independently of each other, have some probability $q = q(n)$ of being positive. This means that the probability distribution on the possible subsets of positives is a binomial distribution, which is a standard assumption in the area of probabilistic group testing (see, e.g., [7, 8, 37]). In this scenario one is interested in minimizing the average number of queries necessary to identify all positives. Shannon's source coding theorem implies that the minimum average number of queries is lower bounded by the entropy

$$n(-q(n)\log q(n) - (1-q(n))\log(1-q(n))). \tag{12}$$

It is also known [7, 37] that for two-stage group testing algorithms there are values of the probability $q(n)$ for which the lower bound (12) is not reachable, in the sense that better lower bounds exist. Our algorithm for the probabilistic case is very simple and is based on the following idea. Given the probability $q = q(n)$ that a single object in $\mathcal{O}$ is positive, we estimate the expected number of positives $\mu = nq(n)$. We now run the two-stage algorithm described in section 3, using a $(k, m, n)$-selector with parameters $m = (1+\delta)\mu + 1$, with $\delta > 0$, and $k = 2(1+\delta)\mu$. Denote by $X$ the random variable taking the value $i$ if and only if the number of positives in $\mathcal{O}$ is exactly $i$. $X$ is distributed according to a binomial distribution with parameter $q$ and mean value $\mu$. If the number of positives is at most $(1+\delta)\mu$, and this happens with probability $Pr[X \leq (1+\delta)\mu]$, then by the result of section 3 the execution of the queries of Stage 1 will restrict our search to $2(1+\delta)\mu - 1$ elements, which will be individually probed during Stage 2. Stage 1 requires $O(m \log \frac{n}{m})$ queries. If, on the contrary, the number of positives is larger than $(1+\delta)\mu$, then the feedback vector $\mathbf{f}$ might cover more than $2(1+\delta)\mu - 1$ columns of the selector. Consequently a larger number of elements, potentially *all* $n$ elements, must be individually probed in Stage 2. The crucial observation is that this latter unfavorable event happens with probability $Pr[X > (1+\delta)\mu]$. Altogether, the above algorithm uses an average number of queries $E$ given by

$$E = O\left(m \log \frac{n}{m}\right) + nPr[X > (1+\delta)\mu]. \tag{13}$$

Choosing $\delta \geq 2e$ and by recalling that $m = (1+\delta)\mu + 1$, we get from (13) and by the Chernoff bound (see [46, p. 72]) that

$$E = O\left(nq(n) \log \frac{1}{q(n)}\right) + n2^{-(1+\delta)nq(n)}. \tag{14}$$

A similar idea was used in [8]. However, the authors of [8] used classical superimposed codes in the first stage of their algorithm, and since these codes have sizes much larger than our selectors, their results are worse than ours. Recalling now the information-theoretic lower bound (12) on the expected number of queries, we get from (14) that our algorithm is provably asymptotically optimal whenever the probability function $q(n)$ satisfies the condition

$$q(n) \geq \frac{1}{n}\left(\log\frac{1}{q(n)} - \log\log\frac{1}{q(n)} - O(1)\right). \tag{15}$$

For instance, $q(n) = c\frac{\log n}{n}$ for any positive constant $c$ or $q(n)$ such that $\frac{q(n)n}{\log n} \to \infty$ satisfy (15). The previous two cases were explicitly considered in [7], where the authors obtained results similar to ours, with better constants. Nevertheless, our condition (15) is more general. Also, our method is rather flexible since one can "tune" the choice of $m$ to some value $f(n)\delta\mu$, for a suitably chosen function $f(n)$, in order to get good performances also when $q(n)$ does not satisfy (15).

The main difference between our results and those of [7] consists of the following. Here we estimate the average number of queries of our explicitly defined algorithm. Instead, the authors of [7] estimate the average number of queries performed by a two-stage algorithm, where the Boolean matrix used in the first stage is randomly chosen among all $m \times n$ binary matrices, where the choice of $m$ depends on $q(n)$. Using a very complex yet accurate analysis, they probabilistically show the *existence* of two-stage algorithms with good performances. For several classes of probability functions $q(n)$, they are able to give asymptotic upper and lower bounds on the minimum average number of queries that differ in several cases only by a multiplicative constant.

## 5. An optimal four-stage group testing algorithm for the GTI model.

In this section we consider the group testing with inhibitors (GTI) model introduced in [29]. We recall that, in this model, in addition to positive items and regular items, there is also a category of items called inhibitors. The inhibitors are the items that interfere with the test by hiding the presence of positive items. As a consequence, a test yields a positive feedback if and only if the tested pool contains one or more positives and no inhibitors. We present an optimal worst-case four-stage group testing algorithm to find $p$ positives in the presence of up to $r$ inhibitors.

**Stage 1.** The goal of this stage is to find a pool $Q \subseteq \mathcal{O}$ which tests positive. To this aim, we associate each item with a distinct column of a $(p, r)$-superimposed code $M = [M(i, j)]$. Let $t$ be the length of the code. For $i = 1, \ldots, t$ we construct the pool $T_i = \{j \in \{1, \ldots, n\} : M(i, j) = 1\}$. If we test pools $T_1, \ldots, T_t$, then the feedback vector has the $i$th entry equal to 1 if and only if at least one the columns associated with the $p$ positives has the $i$th entry equal to 1, whereas none of the columns associated with the inhibitors has the $i$th entry equal to 1. It is easy to prove that such an entry $i$ exists by using the fact that the code $M$ is $(p, r)$-superimposed. Stage 1 returns $Q = T_i$.

**Stage 2.** The goal of this stage is to remove all inhibitors from the set $\mathcal{O}$. To this aim we associate each item not in $Q$ with a distinct column of a $(k', r + 1, n - |Q|)$-selector $M'$. Let $t'$ be the size of the selector. For $i = 1, \ldots, t'$ we construct the pool $T'_i = \{j \in \{1, \ldots, n\} : M'(i, j) = 1\}$. Let $s \leq r$ denote the number of inhibitors in $\mathcal{O}$. If we test pools $T'_1 \cup Q, \ldots, T'_{t'} \cup Q$, then the feedback vector $\mathbf{f}'$ has the $i$th entry equal to 0 if and only if $T'_i$ contains one or more inhibitors. Hence, the feedback vector $\mathbf{f}'$ is equal to the intersection (Boolean product) of the bitwise complement of the $s$ columns associated with the inhibitors. Let $\overline{\mathbf{f}}'$ be the bitwise complement of $\mathbf{f}'$. The column $\overline{\mathbf{f}}'$ is equal to the Boolean sum of the $s$ columns associated with the $s$ inhibitors. Using an argument similar to that used for the two-stage group testing algorithm of section 3, one has that $\overline{\mathbf{f}}'$ covers at most $k' - s - 1$ columns in addition to those associated with the $s$ inhibitor items. We set apart all $k'-1$ items covered by $\overline{\mathbf{f}}'$. These $k'-1$ items will be individually probed in Stage 4 since some of them might be defective.

**Stage 3.** The goal of this stage is to discard a "large" number of regular items from the set of $n-k'$ items remaining after Stage 2. The present stage is similar to Stage 1 of our two-stage algorithm of section 3. We associate each of the $n-k'$ items with a distinct column of a $(k'', p+1, n-k')$-selector $M''$. Let $t''$ be the size of the selector. For $i=1,\dots,t''$ we construct the pool $T''_i = \{j \in \{1,\dots,n\} : M''(i,j)=1\}$ and test pools $T''_1,\dots,T''_{t''}$. Notice that after Stage 2 there is no inhibitor among the searched set of items and, consequently, the feedback vector $\mathbf{f}''$ is equal to the Boolean sum of the columns associated with the positive items in the set (those which have not been set apart in Stage 2). After these $t''$ tests we discard all items but those corresponding to columns covered by the feedback vector $\mathbf{f}''$. Hence, we are left with $k''-1$ items.

**Stage 4.** We individually probe the $k'-1$ items returned by Stage 2 and the $k''-1$ items returned by Stage 3.

The above algorithm provides the following general result.

THEOREM 5. *Let $k'$, $k''$, $n$, $p$, and $r$ be integers with $1 \le r < k' < n$ and $1 \le p < k'' < n-k'$. There exists a four-stage group testing algorithm for finding $p$ positives in the presence of up to $r$ inhibitors by*

$$t_s(p,r,n) + t(k', r+1, n-|Q|) + t(k'', p+1, n-k') + k' + k'' - 2$$

*tests.*

The following main corollary of Theorem 5 holds.

COROLLARY 5. *Let $p$ and $r$ be integers with $1 \le r < n$ and $1 \le p < n-2r$. There exists a four-stage group testing algorithm for finding $p$ positives in the presence of up to $r$ inhibitors by*

$$t_s(p,r,n) + O\left(r\log\frac{n}{r} + p\log\frac{n-r}{p}\right) \tag{16}$$

*tests, and this upper bound is asymptotically optimal.*

*Proof.* By setting $k'=2r$ and $k''=2p$ in Theorem 5 and using the bound of Theorem 1 on the size of selectors, one gets the following upper bound on the number of tests performed by the four-stage algorithm:

$$t_s(p,r,n) + 4er\ln\frac{n-|Q|}{2r} + 2e(4r-1) + 4ep\ln\frac{n-2r}{2p} + 2e(4p-1) + 2r + 2p. \tag{17}$$

We now prove that the above upper bound is asymptotically optimal. In [21] the authors proved the following lower bound of

$$\Omega\left(t_s(p,r,n-p-1) + \ln\binom{n}{p}\right) \tag{18}$$

on the number of tests required by *any* algorithm (using any number of stages) to find $p$ defectives in the presence of $r$ inhibitors. It is easy to see that any $(p,r)$-superimposed code of size $n-p-1$ and length $t$ can be transformed into a $(p,r)$-superimposed code of size $n$ and length $t+p+1$. Indeed, let $M$ be a $(p,r)$-superimposed code of size $n-p-1$ and length $t$, and let $c_j$, $j=1,\dots,p+1$, denote the binary column of length $t+p+1$ with all entries, except that in position $t+j$, equal to zero. If we add $p+1$ entries equal to zero at the end of each column of $M$ and introduce the columns $c_1, \dots, c_{p+1}$ in the resulting code, then we obtain a $(p,r)$-superimposed code of size $n$ and length $t+p+1$. It follows that $t_s(p,r,n-p-1)+p+1 = \Omega(t_s(p,r,n))$ and, consequently, the lower bound (18) is

$$\Omega\left(t_s(p,r,n) + \ln\binom{n}{p}\right). \tag{19}$$

It is possible to see that expression (19) is $\Omega(t_s(p,r,n) + r\log\frac{n}{r} + p\log\frac{n}{p})$. If $p > r$, this is immediate. If $p \le r$, Theorem 2 implies that

$$t_s(p,r,n) = \Omega\left(r\log\frac{n}{r}\right). \tag{20}$$

Therefore expression (19) is $\Omega(t_s(p,r,n) + r\log\frac{n}{r} + p\log\frac{n}{p})$. It follows that the upper bound (17) on the number of tests performed by the four-stage algorithm is tight with lower bound (18). $\square$

We can employ a $(p+r, r+1, n)$-selector in Stage 1 of the four-stage algorithm and use the bound of Theorem 1 on the size of selectors to estimate the number of tests performed by this stage. Notice that the weight of the rows of the $(p+r, r+1, n)$-selector corresponds to the size of the pools tested during Stage 1 and, consequently, to that of the set $Q$ returned by this stage. By using the construction of Theorem 1 one has that the size of $Q$ is $\frac{n}{r+p}$. Hence, the following result holds.

Corollary 6. *For any integers $p, r$, and $n$, with $p \ge 1$, $r \ge 0$ and $p + r \le n$, there exists a four-stage group testing algorithm for finding $p$ positives in a set of $n$ elements, up to $r$ of which can be inhibitors, using a number of tests at most*

$$\frac{e(p+r)^2}{p}\ln\frac{n}{p+r} + 4er\ln\frac{n(r+p-1)}{2r(r+p)} + 4ep\ln\frac{n-2r}{2p}$$
$$+ (10e+2)p + (12e+2)r - 5e + \frac{er(2r-1)}{p} - 2.$$

It is remarkable that for $r = O(p)$ Corollary 6 implies that our deterministic algorithm attains the same asymptotic complexity $O((r+p)\log n)$ of the randomized algorithm presented in [29].

Notice that the algorithm presented in this section actually discovers both the positives *and* the inhibitors. Therefore, in the same spirit of section 3.2, we observe that the problem of finding $p$ positives and $r$ inhibitors is equivalent to the problem of learning an unknown Boolean function of the form $(x_1 \vee \cdots \vee x_p) \wedge \overline{(y_1 \vee \cdots \vee y_r)}$. Hence, the above results can be rephrased as follows.

Corollary 7. *For any $p$ and $r$, Boolean functions of the form $(x_1 \vee \cdots \vee x_p) \wedge \overline{(y_1 \vee \cdots \vee y_s)}$, $s \le r$, are exactly learnable in only* four stages *by using a number of queries of the same order as that of the best fully adaptive learning algorithm.*

## 6. Final discussion.

The main result obtained in this paper is the first two-stage group testing algorithm that uses a number of tests of the same order as the information-theoretic lower bound on the problem. In retrospect, it could be useful to see how we have reached our goal. The first stage of our algorithm is conceptually similar to a classical totally nonadaptive group testing algorithm that first encodes the items of the search space $\mathcal{O}$ with the column vectors of a superimposed code and then performs tests according to the subsets of $\mathcal{O}$ specified by the rows of the matrix constituting the code (recall section 3). Unfortunately, any matrix that represents a superimposed code has $\Omega((p^2/\log p)\log n)$ number of rows, where $p$ is the known upper bound on the number of positive elements, and therefore the number of performed tests is well above the information-theoretic lower bound $\log\binom{n}{p}$. Hence, we are lead to look for combinatorial objects satisfying weaker conditions and having a number of rows at most $O(p\log n)$ so that, by using them to specify the tests of the first stage, we would remain close to the information-theoretic lower bound. Of course, if we used such objects, whatever they are, at the end of the first stage we would necessarily have unclassified items (recall the lower bound of $\Omega((p^2/\log p)\log n)$ on the number of tests for one-stage algorithms that correctly discriminate all items in positives and negatives). Luckily, by using our selectors of length $O(\log\binom{n}{p})$, at the end of the first stage the number of unclassified items is very small, at most $2p-1$, and by testing them one by one we find all positives, and we remain asymptotically within the border of the information-theoretic lower bound. We recall that our algorithm uses an overall number of tests that is less than $7p\log\frac{n}{p}+O(p)$; therefore the relevant constant in $O(p\log(n/p))$ is also reasonably small.

It would be interesting to see whether this approach of "weakening" combinatorial structures that are used in totally nonadaptive search procedures could be used also for other problems to obtain "few-stages" algorithms, with the same asymptotic performances of totally adaptive algorithms. The general approach should be the following: since it may be too expensive to determine in a totally nonadaptive fashion the exact solution to the problem, one first nonadaptively individuates an "approximate" solution to the problem, that is, a small set of potential solutions, and thereafter one searches for the exact solution in this small set.

There are numerous problems that are possible candidates for such an investigation; potentially, many of the search problems mentioned in [2, 24], for which there exists a gap between the complexity of adaptive and nonadaptive algorithms, could be studied in this light. A particularly interesting one is the so-called "group testing for complexes" [26, 44]. Here the elements of $\mathcal{O}$ are positive or negative not by themselves, but only in conjunction with some others (imagine some chemical substances that react with others and therefore cannot be mixed with them, but yet are nonreacting with other, different substances). The problem is to identify all positive subsets, that is, to remain in the chemical scenario, to identify all groups of mutually reacting substances. The best nonadaptive algorithm for the above problem has huge complexity [26], and an interesting open problem would be to see whether our approach could lead to efficient "few-stage" algorithms.

In general, our study raises the following question: How much adaptiveness is really needed in search procedures to obtain the optimal performances of fully adaptive algorithms? Our results show that in group testing one needs to use adaptiveness only once. There are other situations in which this phenomenon occurs. For instance, for the well-known Renyi–Ulam game [48, 50, 53], in which one looks for an unknown number by asking arbitrary yes/no questions, at most a fixed number of which can receive an erroneous answer, the authors of [14] have shown that there exist two-stage search strategies using a number of questions exactly equal to the information-theoretic lower bound, and that one-stage search strategies cannot reach this lower bound. Conversely, it is also known that there are natural search problems for which fully adaptive algorithms are better than any $k$-stage algorithm (see, e.g., [47]). A better understanding of the above issues is worth pursuing.

## REFERENCES

[1] R. Ahlswede and I. Wegener, *Search Problems*, John Wiley & Sons, New York, 1987.

[2] M. Aigner, *Combinatorial Search*, Wiley–Teubner, New York, Stuttgart, 1988.

[3] N. Alon and V. Asodi, *Learning a hidden subgraph*, SIAM J. Discrete Math., 18 (2005), pp. 697–712.

[4] D. Angluin, *Queries and concept learning*, Mach. Learn., 2 (1987), pp. 319–342.

[5] E. Barillot, B. Lacroix, and D. Cohen, *Theoretical analysis of library screening using an n-dimensional pooling strategy*, Nucleic Acids Res., 19 (1991), pp. 6241–6247.

[6] D. J. Balding, W. J. Bruno, E. Knill, and D. C. Torney, *A comparative survey of non-adaptive pooling design*, in Genetic Mapping and DNA Sequencing, IMA Vol. Math. Appl. 81, T. P. Speed and M. S. Waterman, eds., Springer-Verlag, Berlin, 1996, pp. 133–154.

[7] T. Berger and V. I. Levenshtein, *Asymptotic efficiency of two-stage disjunctive testing*, IEEE Trans. Inform. Theory, 48 (2002), pp. 1741–1749.

[8] T. Berger and V. I. Levenshtein, *Application of cover-free codes and combinatorial design to two-stage testing*, Discrete Appl. Math., 128 (2003), pp. 11–26.

[9] T. Berger, J. W. Mandell, and P. Subrahmanya, *Maximally efficient two-stage screening*, Biometrics, 56 (2000), pp. 833–840.

[10] W. J. Bruno, D. J. Balding, E. Knill, D. Bruce, C. Whittaker, N. Dogget, R. Stalling, and D. C. Torney, *Design of efficient pooling experiments*, Genomics, 26 (1995), pp. 21–30.

[11] P. Bussbach, *Constructive Methods to Solve Problems of s-surjectivity, Conflict Resolution, and Coding in Defective Memories*, Tech. report 84D005, Ecole Nationale des Telecomm., ENST Paris, 1984.

[12] S. Chaudhuri and J. Radhakrishnan, *Deterministic restrictions in circuit complexity*, in Proceedings of the Twenty-Eighth Annual ACM Symposium on the Theory of Computing (STOC '96), ACM, New York, 1996, pp. 30–36.

[13] M. Chrobak, L. Gąsieniec, and W. Rytter, *Fast broadcasting and gossiping in radio networks*, J. Algorithms, 43 (2002), pp. 177–189.

[14] F. Cicalese, D. Mundici, and U. Vaccaro, *Least adaptive optimal search with unreliable tests*, Theoret. Comput. Sci., 270 (2002), pp. 877–893.

[15] A. E. F. Clementi, A. Monti, and R. Silvestri, *Selective families, superimposed codes, and broadcasting on unknown radio networks*, in Proceedings of the Twelth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'01), SIAM, Philadelphia, ACM, New York, 2001, pp. 709–718.

[16] G. Cormode and S. Muthukrishnan, *What's hot and what's not: Tracking most frequent items dynamically*, in Proceedings of the 22nd ACM Symposium on Principles of Database Systems, ACM, New York, 2003, pp. 296–306.

[17] P. Damaschke, *Adaptive versus nonadaptive attribute-efficient learning*, in Proceedings of the Thirtieth Annual ACM Symposium on Theory of Computing (STOC '98), ACM, New York, 1998, pp. 590–596.

[18] P. Damaschke, *Parallel attribute-efficient learning of monotone Boolean functions*, in Algorithm Theory—SWAT 2000, Lecture Notes in Comput. Sci. 1851, M. M. Halldórsson, ed., Springer-Verlag, Berlin, 2000, pp. 504–512.

[19] P. Damaschke, *Computational aspects of parallel attribute-efficient learning*, in Proceedings of the 9th International Conference on Algorithmic Learning Theory, Lecture Notes in Comput. Sci. 1501, M. Richter et al., eds., Springer-Verlag, Berlin, 1998, pp. 103–111.

[20] P. Damaschke, *Randomized group testing for mutually obscuring defectives*, Inform. Process. Lett., 67 (1998), pp. 131–135.

[21] A. De Bonis and U. Vaccaro, *Improved algorithms for group testing with inhibitors*, Inform. Process. Lett., 66 (1998), pp. 57–64.

[22] A. De Bonis and U. Vaccaro, *Constructions of generalized superimposed codes with applications to group testing and conflict resolution in multiple access channels*, Theoret. Comput. Sci., 306 (2003), pp. 223–243.

[23] R. Dorfman, *The detection of defective members of large populations*, Ann. Math. Statist., 14 (1943), pp. 436–440.

[24] D. Z. Du and F. K. Hwang, *Combinatorial Group Testing and Its Applications*, World Scientific, River Edge, NJ, 2000.

[25] A. G. Dyachkov and V. V. Rykov, *A survey of superimposed code theory*, Problems Control Inform. Theory, 12 (1983), pp. 229–242.

[26] A. G. Dyachkov, P. Vilenkin, A. Macula, and D. Torney, *Families of finite sets in which no intersection of $\ell$ sets is covered by the union of $s$ others*, J. Combin. Theory Ser. A, 99 (2002), pp. 195–218.
[27] A. G. Dyachkov, *Private communication*, 2003.
[28] P. Erdös, P. Frankl, and Z. Füredi, *Families of finite sets in which no set is covered by the union of $r$ others*, Israel J. Math., 51 (1985), pp. 75–89.
[29] M. Farach, S. Kannan, E. H. Knill, and S. Muthukrishnan, *Group testing problems with sequences in experimental molecular biology*, in Proceedings of Compression and Complexity of Sequences, B. Carpentieri, A. De Santis, U. Vaccaro, and J. Storer, eds., IEEE, Piscataway, NJ, 1997, pp. 357–367.
[30] Z. Füredi, *On $r$-cover free families*, J. Combin. Theory Ser. A, 73 (1996), pp. 172–173.
[31] E. H. Hong and R. E. Ladner, *Group testing for image compression*, IEEE Trans. Image Process., 11 (2002), pp. 901–911.
[32] H. Q. Ngo and D.-Z. Du, *A survey on combinatorial group testing algorithms with applications to DNA library screening*, in Discrete Mathematical Problems with Medical Applications, DIMACS Ser. Discrete Math. Theoret. Comput. Sci. 55, AMS, Providence, RI, 2000, pp. 171–182.
[33] F. K. Hwang and Y. C. Liu, *Error-tolerant pooling designs with inhibitors*, J. Comput. Biol., 10 (2003), pp. 231–236.
[34] P. Indyk, *Deterministic superimposed coding with application to pattern matching*, in Proceedings of the Thirty-ninth IEEE Annual Symposium on Foundations of Computer Science (FOCS '97), IEEE, Piscataway, NJ, 1997, pp. 127–136.
[35] P. Indyk, *Explicit constructions of selectors and related combinatorial structures, with applications*, in Proceedings of the Thirteenth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'02), SIAM, Philadelphia, ACM, New York, 2002, pp. 697–704.
[36] W. H. Kautz and R. R. Singleton, *Nonrandom binary superimposed codes*, IEEE Trans. Inform. Theory, 10 (1964), pp. 363–377.
[37] E. Knill, *Lower bounds for identifying subset members with subset queries*, in Proceedings of the Sixth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA '95), SIAM, Philadelphia, ACM, New York, 1995, pp. 369–377.
[38] J. Komlós and A. G. Greenberg, *An asymptotically fast non-adaptive algorithm for conflict resolution in multiple-access channels*, IEEE Trans. Inform. Theory, 31 (1985), pp. 302–306.
[39] R. Kumar, S. Rajagopalan, and A. Sahai, *Coding constructions for blacklisting problems without computational assumptions*, in Proceedings of the 19th Annual International Cryptology Conference (CRYPTO '99), Lecture Notes in Comput. Sci. 1666, Springer-Verlag, Berlin, 1999, pp. 609–623.
[40] C. H. Li, *A sequential method for screening experimental variables*, J. Amer. Statist. Assoc., 57 (1962), pp. 455–477.
[41] N. Linial, *Locality in distributed graph algorithms*, SIAM J. Comput., 21 (1992), pp. 193–201.
[42] L. Lovàsz, *On the ratio of optimal integral and fractional covers*, Discrete Math., 13 (1975), pp. 383–390.
[43] A. J. Macula, *Probabilistic nonadaptive and two-stage group testing with relatively small pools and DNA library screening*, J. Comb. Optim., 2 (1999), pp. 385–397.
[44] A. J. Macula, P. Vilenkin, and D. Torney, *Two-stage group testing for complexes in the presence of errors*, in Discrete Mathematical Problems with Medical Applications, DIMACS Ser. Discrete Math. Theoret. Comput. Sci. 55, AMS, Providence, RI, 2000, pp. 145–157.
[45] D. Margaritis and S. Skiena, *Reconstructing strings from substrings in rounds*, in Proceedings of the Thirty-seventh IEEE Annual Symposium on Foundations of Computer Science (FOCS '95), IEEE, Piscataway, NJ, 1995, pp. 613–620.
[46] R. Motwani and P. Raghavan, *Randomized Algorithms*, Cambridge University Press, Cambridge, UK, 1995.
[47] A. Pelc, *Weakly adaptive comparison searching*, Theoret. Comput. Sci., 66 (1989), pp. 105–111.
[48] A. Pelc, *Searching games with errors—fifty years of coping with liars*, Theoret. Comput. Sci., 243 (2002), pp. 71–109.
[49] P. A. Pevzner and R. J. Lipshutz, *Towards DNA sequencing chips*, in Proceedings of the 19th International Symposium on Mathematical Foundations of Computer Science, Lecture Notes in Comput. Sci. 841, Springer-Verlag, Berlin, 1994, pp. 143–158.
[50] R. L. Rivest, A. R. Meyer, D. J. Kleitman, K. Winklmann, and J. Spencer, *Coping with errors in binary search procedures*, J. Comput. System Sci., 20 (1980), pp. 396–404.

[51] M. Ruszinkó, *On the upper bound of the size of the $r$-cover-free families*, J. Combin. Theory Ser. A, 66 (1994), pp. 302–310.
[52] M. Sobel and P. A. Groll, *Group testing to eliminate efficiently all defectives in a binomial sample*, Bell System Tech. J., 38 (1959), pp. 1179–1252.
[53] J. Spencer, *Ulam's searching game with a fixed number of lies*, Theoret. Comput. Sci., 95 (1992), pp. 307–321.
[54] D. R. Stinson, T. van Trung, and R. Wei, *Secure frameproof codes, key distribution patterns, group testing algorithms and related structures*, J. Statist. Plann. Inference, 86 (2000), pp. 595–617.
[55] J. Wolf, *Born again group testing: Multiaccess communications*, IEEE Trans. Inform. Theory, 31 (1985), pp. 185–191.
[56] M. Xie, K. Tatsuoka, J. Sacks, and S. S. Young, *Group testing with blockers and synergism*, J. Amer. Statist. Assoc., 96 (2001), pp. 92–102.

SIAM J. COMPUT. © 2005 Society for Industrial and Applied Mathematics
Vol. 34, No. 5, pp. 1271–1278

# STRICTLY NONBLOCKING MULTIRATE $\log_d(N, m, p)$ NETWORKS*

FRANK K. HWANG†, YONG HE‡, AND YANG WANG‡

**Abstract.** We give necessary and sufficient conditions for the $d$-nary multilog network to be strictly nonblocking under the discrete multirate model, and we give sufficient conditions for the same under the continuous multirate model.

**Key words.** strictly nonblocking network, multirate model, multilog network, Cantor network

**AMS subject classifications.** 68M10, 90B18

**DOI.** 10.1137/S0097539703433158

## 1. Introduction.

In a multirate network, each link has a (normalized) capacity 1 and each request for connection is associated with a weight (bandwidth requirement) $w$. Many paths can go through a link simultaneously as long as their total weight has not exceeded unity. In particular, an input or output (link) can generate or receive many requests as long as their total weight does not exceed unity. Often, the weight of a request is bounded in the range $[b, B]$. A more general model is to assume that an input or output link has capacity $\beta \leq 1$ to reflect the reality that many networks need an internal-to-external speed-up to be more efficient. In the discrete case (the channel model), we assume that each internal link has $f_1$ channels, each input or output has $f_0 \leq f_1$ channels, and a request is associated with a positive integer number $q, 1 \leq q \leq Q$, where $Q \leq f_0$ is an upper bound of the number of channels a request can demand.

A network state is a set of paths connecting a set of requests $\{(i_x, o_y, w)\}$ such that no link carries a load exceeding 1 (or $f_1$), where $i_x$ is an input, $o_y$ is an output, and $w$ is the associated weight. Given a state, a new request $(i, o, w)$ must satisfy the condition that $i$ has not generated and $o$ has not received requests whose total weight exceeds $1 - w$ (or $f_0 - w$). A network is strictly nonblocking if, at any state, a new request can always be connected without having any links carrying a load exceeding 1 (or $f_1$).

Strictly nonblocking multirate networks have been studied for the 3-stage Clos network [3] and the $d$-nary Cantor network [1], $d \geq 2$. In this paper, we extend the results of the Cantor network to the more general $\log_d(N, m, p)$ network (also called the $d$-nary multilog network), where the Cantor network is the special case with $m = n - 1$. In particular, we give necessary and sufficient conditions for $\log_d(N, m, p)$ to be multirate strictly nonblocking.

*Received by the editors August 14, 2003; accepted for publication (in revised form) March 4, 2005; published electronically June 16, 2005. The research of the first author was done during a visit to the Center of Mathematical Sciences at Zhejiang University. The research of the second author was supported by National Natural Science Foundation of China (10271110, 60021201) and by TRAPOYT of MOE, China.

http://www.siam.org/journals/sicomp/34-5/43315.html

†Department of Applied Mathematics, National Jiaotung University, Hsinchu, Taiwan, 30050, ROC (fhwang@math.nctu.edu.tw).

‡Department of Mathematics, Zhejiang University, Hangzhou 310027, P. R. China (mathhey@zju.edu.cn, wangyang_zju@yahoo.com.cn).

**2. The channel model.** The $\log_d(N, m, p)$ network was first proposed by Shyy and Lea [6], extending the $\log_d(N, 0, p)$ network proposed by Lea [4]. The $\log_d(N, m, p)$ network has an input (output) stage consisting of $N = d^n$ $1 \times p$ ($p \times 1$) crossbars and $p$ copies of the $d$-nary $m$-extra-stage, $0 \leq m \leq n-1$, inverse banyan network $BY_d^{-1}(n, m)$, where each input and output crossbar is connected to every copy of $BY_d^{-1}(n, m)$. We will label the $n + m + 2$ stages by $0, 1, \dots, n + m + 1$. Figure 1 illustrates an example of $\log_d(N, m, p)$.

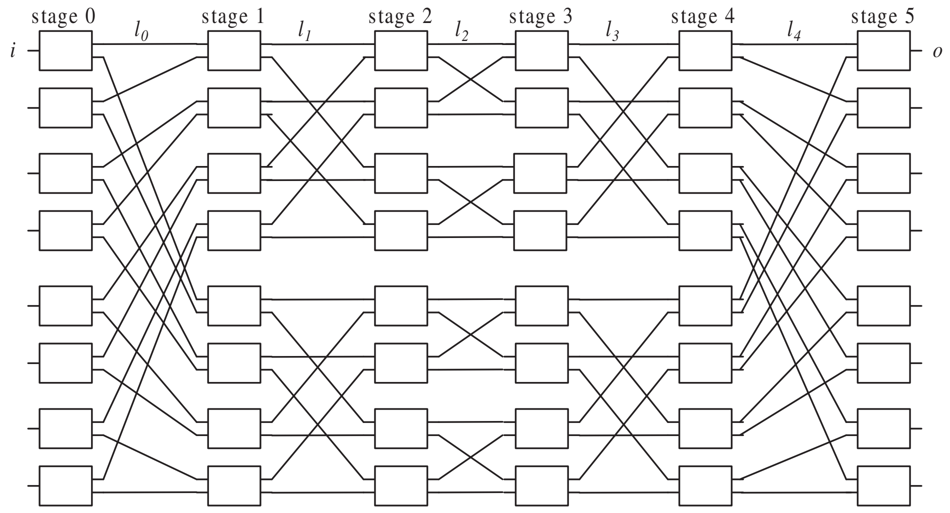


Fig. 1. *A* $\log_2(8, 1, 2)$ *network.*

Next we study the channel model in this section.

Theorem 2.1. *Consider the* $(Q, f_0, f_1)$ *channel model with* $d^{\lfloor \frac{n-1}{2} \rfloor} f_0 \geq f_1 + 1$. *Then* $\log_d(N, 0, p)$ *is multirate strictly nonblocking if and only if*

$$p \geq \left\lfloor \frac{d^{\lfloor \frac{n-1}{2} \rfloor} f_0 - Q}{f_1 - Q + 1} \right\rfloor + \left\lfloor \frac{d^{\lceil \frac{n-1}{2} \rceil} f_0 - Q}{f_1 - Q + 1} \right\rfloor + 1.$$

*Proof.* Suppose the new request is $(i, o, q)$. Call an internal link *$q$-saturated* if it carries a load exceeding $f_1 - q + 1$ (and hence cannot carry a new $q$-request). Note that the channel graph between $i$ and $o$ is just a single path $l$. An intersecting path is a path $p$ from input $i' \neq i$ to output $o' \neq o$ such that $p$ shares a link with $l$. In particular, let $l_j$ denote the stage-$j$ link (a link between stage $j$ and stage $j + 1$) in $l$ (see Figure 1). Then a $j$-intersecting path is one which intersects $l$ at $l_j$. Note that a $j$-intersecting path can also be a $j'$-intersecting path for $j \neq j'$. Further, $l_0$ ($l_n$) cannot be saturated since its load is from $i$ ($o$). Therefore we need only look inside the $BY_d^{-1}(n, 0)$ for saturated links.

*Sufficiency.* The new request cannot be carried in a copy of $BY_d^{-1}(n, 0)$ if and only if there exists an $l_j$ which is $q$-saturated by $j$-intersecting paths. We divide $l$ into two disjoint halves:

$$H_1 = \{l_j : 1 \leq j \leq \lfloor (n-1)/2 \rfloor\},$$

$$H_2 = \{l_j : \lfloor (n+1)/2 \rfloor \leq j \leq n-1\}.$$

Note that only $d^{\lfloor (n-1)/2 \rfloor}$ inputs can generate $j$-intersecting paths over all $j$ in $H_1$. The total weight of these paths is bounded by $d^{\lfloor (n-1)/2 \rfloor} f_0 - q$ since the weight of the new request must be excluded. In the right-hand side of the inequality in Theorem 2.1, the first term is an upper bound of the number of $q$-saturated $l_j$ over all $j$ in $H_1$. Similarly, only $d^{\lceil (n-1)/2 \rceil}$ outputs can generate $j$-intersecting paths for $j$ in $H_2$, and the second term in the inequality is an upper bound of the number of $q$-saturated $l_j$ over all $j$ in $H_2$. Thus their sum is an upper bound of the number of saturated links in $l$, and hence an upper bound of the number of blocked copies of $BY_d^{-1}(n,0)$. One more copy suffices to route the new request.

*Necessity.* It suffices to construct a set of connections intersecting the $(i,o,Q)$ request with total weights of $\lfloor \frac{d^{\lfloor \frac{n-1}{2} \rfloor} f_0 - Q}{f_1 - Q + 1} \rfloor$ and $\lfloor \frac{d^{\lceil \frac{n-1}{2} \rceil} f_0 - Q}{f_1 - Q + 1} \rfloor$. With respect to $l$, an input (output) is called *$j$-marginal*, $1 \le j \le n-1$, if it can generate a $j$-intersecting path but not a $(j-1)$-intersecting ($(j+1)$-intersecting) path. Let $WI_j$ ($WO_j$) denote the total weight of requests which can be generated by $j$-marginal inputs (outputs). Then

$$WI_1 = df_0 - q, \quad WI_j = (d^j - d^{j-1}) f_0 \quad \text{for} \ \ 2 \le j \le n-1,$$

$$WO_j = WI_{n-j} \quad \text{for} \ \ 1 \le j \le n-1.$$

Note that

$$WI_j < WO_j \quad \text{for} \ \ j \in H_1$$

and

$$WI_j > WO_j \quad \text{for} \ \ j \in H_2.$$

Compute the maximum number $b_1$ of $Q$-saturated links generated by 1-marginal inputs. Assign a total weight of $b_1(f_1 - Q + 1)$ of requests to 1-marginal outputs (doable by the above inequalities), and mix the remaining requests of weight $df_0 - Q - b_1(f_1 - Q + 1)$ with requests generated by 2-marginal inputs. Again, compute the maximum number $b_2$ of $Q$-saturated links generated by this mixture of requests. Assign requests with a total weight $b_2(f_1 - Q + 1)$ to 2-marginal outputs, and mix the rest with requests from 3-marginal inputs. Proceed like this until the last step $s = \lfloor (n-1)/2 \rfloor$. At step $s$, assign requests with a total weight of $b_s(f_1 - Q + 1)$ to $s$-marginal outputs and ignore unassigned requests. It is straightforward to verify that the number of saturated $l_j$ for $j$ in $H_1$ constructed by this assignment is the first term in the inequality of Theorem 2.1. Similarly, the corresponding number for $j$ in $H_2$ is the second term. Thus their sum plus one copy is the necessary number of copies needed to route the new request. $\square$

Next we consider the general $m$ case. Define

$$g_m(q) = \sum_{j=1}^{m} \frac{1}{d^j} \left\{ \left\lfloor \frac{d^j f_0 - q}{f_1 - q + 1} \right\rfloor - \left\lfloor \frac{d^{j-1} f_0 - q}{f_1 - q + 1} \right\rfloor \right\} \qquad \text{for} \quad 0 \le m \le n-1.$$

THEOREM 2.2. *Consider the $(Q, f_0, f_1)$ channel model with $d^{\lfloor \frac{n-1}{2} \rfloor} f_0 \ge f_1 + 1$. Then $\log_d(N,m,p)$ is multirate strictly nonblocking for $0 \le m \le n-1$ if and only if*

$$p \geq \left\lfloor 2g_m(Q) + \left\lfloor \frac{d^{\lfloor \frac{n+m-1}{2} \rfloor} f_0 - Q - \lfloor \frac{d^m f_0 - Q}{f_1 - Q + 1} \rfloor (f_1 - Q + 1)}{f_1 - Q + 1} \right\rfloor \frac{1}{d^m} \right.$$

$$\left. + \left\lfloor \frac{d^{\lceil \frac{n+m-1}{2} \rceil} f_0 - Q - \lfloor \frac{d^m f_0 - Q}{f_1 - Q + 1} \rfloor (f_1 - Q + 1)}{f_1 - Q + 1} \right\rfloor \frac{1}{d^m} \right\rfloor + 1.$$

*Proof.* Our strategy is to partition the $BY_d^{-1}(n, m)$ stages into two parts as follows: the outer part consists of $m + 1$ outer stages from the input side and $m + 1$ outer stages from the output side; and the inner part consists of $n - m$ inner stages, i.e., stage $m + 1$ to stage $n$, composed of $d^m$ copies of $BY_d^{-1}(n - m, 0)$. (Note that stage $m + 1$ ($n$) is in both parts.) For the outer part, we adopt (and extend) the approach of Chung and Ross [2] given for the special case of the Cantor network. For the inner part we apply Theorem 2.1.

More specifically, for the outer part, we compute the total weight of requests which can reach a stage-$j$ link, $1 \leq j \leq m$, in the $(i, o)$ channel graph that is $d^j f_0 - q$, while a $q$-saturated link carries a load of at least $f_1 - q + 1$. Thus, at most

$$\left\lfloor \frac{d^j f_0 - q}{f_1 - q + 1} \right\rfloor$$

links in the channel graph at or before stage $j$ can be saturated. The worst case is to assign the saturated links to as early a stage as possible since links in the early stages have more blocking power. This results in assigning

$$\left\lfloor \frac{d^j f_0 - q}{f_1 - q + 1} \right\rfloor$$

saturated links to stage $j$, each of which blocks $1/d^j$ copies of $BY_d^{-1}(n, m)$. Thus $\lfloor g_m(q) \rfloor$ is the number of copies of $BY_d^{-1}(n, m)$ blocked by paths intersecting the links of the $(i, o)$ channel graph in the first or last $m$ stages.

The total weight of requests which can reach a stage-$\lfloor (n + m - 1)/2 \rfloor$ link is

$$d^{\lfloor (n+m-1)/2 \rfloor} f_0 - q.$$

But a total weight of

$$\left\lfloor \frac{d^m f_0 - q}{f_1 - q + 1} \right\rfloor (f_1 - q + 1)$$

was already connected in the first $m$ stages. Therefore only the difference of these two weights can be used to saturate stage-$j$ links for $m+1 < j \leq \lfloor (n+m-1)/2 \rfloor$. Since the $\log_d(N, m, p)$ network between these stages consists of $d^m$ copies of $BY_d^{-1}(n - m, 0)$, we apply Theorem 2.1 (only the input side) to the number of copies of $BY_d^{-1}(n - m, 0)$ blocked, which must be divided by $d^m$ to convert to the number of copies of $BY_d^{-1}(n, m)$ blocked.

The argument for the output side is analogous. One extra channel then guarantees the routing of the current request. Therefore

$$p \geq \max_{1 \leq q \leq Q} \left\{ \lfloor 2g_m(q) + \left\lfloor \frac{d^{\lfloor \frac{n+m-1}{2} \rfloor} f_0 - q - \lfloor \frac{d^m f_0 - q}{f_1 - q + 1} \rfloor (f_1 - q + 1)}{f_1 - q + 1} \right\rfloor \frac{1}{d^m} \right.$$

$$\left. \left\lfloor \frac{d^{\lceil \frac{n+m-1}{2} \rceil} f_0 - q - \lfloor \frac{d^m f_0 - q}{f_1 - q + 1} \rfloor (f_1 - q + 1)}{f_1 - q + 1} \right\rfloor \frac{1}{d^m} \rfloor \right\} + 1 \tag{1}$$

is a sufficient condition for $\log_d(N,m,p)$ to be multirate strictly nonblocking for $0 \leq m \leq n-1$. We show that the maximum is achieved at $q = Q$. To see this, define

$$A_j = \left\lfloor \frac{d^j f_0 - q}{f_1 - q + 1} \right\rfloor, \qquad j = 0, 1, \ldots, \left\lfloor \frac{n+m-1}{2} \right\rfloor$$

for simplicity. Then, trivially, $A_0 = \lfloor \frac{f_0 - q}{f_1 - q + 1} \rfloor = 0$ and $A_j \geq 0$ is nondecreasing in $q$ for every $j = 0, 1, \ldots, \lfloor \frac{n+m-1}{2} \rfloor$. We have

$$\begin{aligned}
&g_m(q) + \left\lfloor \frac{d^{\lfloor \frac{n+m-1}{2} \rfloor} f_0 - q - \lfloor \frac{d^m f_0 - q}{f_1 - q + 1} \rfloor (f_1 - q + 1)}{f_1 - q + 1} \right\rfloor \frac{1}{d^m} \\
&= g_m(q) + \left( \left\lfloor \frac{d^{\lfloor \frac{n+m-1}{2} \rfloor} f_0 - q}{f_1 - q + 1} \right\rfloor - \left\lfloor \frac{d^m f_0 - q}{f_1 - q + 1} \right\rfloor \right) \frac{1}{d^m} \\
&= \frac{1}{d}(A_1 - A_0) + \frac{1}{d^2}(A_2 - A_1) + \cdots + \frac{1}{d^m}(A_m - A_{m-1}) + \frac{1}{d^m}(A_{\lfloor \frac{n+m-1}{2} \rfloor} - A_m) \\
&= \left( \frac{1}{d} - \frac{1}{d^2} \right) A_1 + \left( \frac{1}{d^2} - \frac{1}{d^3} \right) A_2 + \cdots + \left( \frac{1}{d^{m-1}} - \frac{1}{d^m} \right) A_{m-1} + \frac{1}{d^m} A_{\lfloor \frac{n+m-1}{2} \rfloor}.
\end{aligned}$$

Since every term is positive and nondecreasing in $q$, we conclude that

$$g_m(q) + \left\lfloor \frac{d^{\lfloor \frac{n+m-1}{2} \rfloor} f_0 - q - \lfloor \frac{d^m f_0 - q}{f_1 - q + 1} \rfloor (f_1 - q + 1)}{f_1 - q + 1} \right\rfloor \frac{1}{d^m}$$

is maximized at $Q$. A similar conclusion holds for the term

$$g_m(q) + \left\lfloor \frac{d^{\lceil \frac{n+m-1}{2} \rceil} f_0 - q - \lfloor \frac{d^m f_0 - q}{f_1 - q + 1} \rfloor (f_1 - q + 1)}{f_1 - q + 1} \right\rfloor \frac{1}{d^m}.$$

It follows that (1) is maximized at $Q$.

The necessity part follows from the fact that both the conditions of Chung and Ross and those of Theorem 2.1 are necessary. However, the $j$-marginal input is defined only for $1 \leq j \leq n$ since they use up all inputs. Similarly, the $j$-marginal output is defined only for $n \leq j \leq n+m-1$, and $WO_j = WI_{n+m-j}$. Therefore we need to slightly modify the assignment of requests from $j$-marginal inputs and outputs. It is easily verified as follows:

$$\sum_{j=1}^{m} WI_j < WO_m \quad \text{and} \quad \sum_{j=n}^{n+m-1} WO_j < WI_n.$$

The modification is to assign requests from $j$-marginal inputs for all $1 \leq j \leq m$ to $m$-marginal outputs and requests for $j$-marginal outputs for all $n \leq j \leq n+m-1$ to $n$-marginal inputs. The above inequalities ensure that such assignments are doable. $\square$

Note that for $m = 0$, $g_m(Q) = 0$, and $\lfloor \frac{d^m f_0 - Q}{f_1 - Q + 1} \rfloor = 0$, Theorem 2.2 is reduced to Theorem 2.1. For $m = n-1$, the terms

$$\left\lfloor \frac{d^{\lfloor \frac{n+m-1}{2} \rfloor} f_0 - Q - \lfloor \frac{d^m f_0 - Q}{f_1 - Q + 1} \rfloor (f_1 - Q + 1)}{f_1 - Q + 1} \right\rfloor$$

and

$$\left\lfloor \frac{d^{\lceil \frac{n+m-1}{2} \rceil} f_0 - Q - \lfloor \frac{d^m f_0 - Q}{f_1 - Q + 1} \rfloor (f_1 - Q + 1)}{f_1 - Q + 1} \right\rfloor$$

equal 0. Theorem 2.2 is reduced to the result on the Cantor network in [2].

We further study the situation when the internal links have different capacities. Suppose the input and output have capacity $f_0$, the stage-$j$ links and stage-$(n-j)$ links have capacity $f_j$, $j = 1, 2, \ldots, \lceil \frac{n+m-1}{2} \rceil$, and $f_{j-1} \le f_j$.

We define

$$l_k(q) = \left\lfloor \frac{d^k f_0 - q - \sum_{i=1}^{k-1} (f_i - q + 1) l_i(q)}{f_k - q + 1} \right\rfloor \qquad \text{for} \quad 1 \le k \le \left\lceil \frac{n+m-1}{2} \right\rceil,$$

and

$$g_m(q) = \sum_{j=1}^{m} \frac{1}{d^k} l_k(q) \qquad \text{for} \quad 0 \le m \le n-1.$$

THEOREM 2.3. *Consider the* $(Q, f_0, f_1, \ldots, f_{\lceil \frac{n+m-1}{2} \rceil})$ *channel model with* $d^{\lfloor \frac{n-1}{2} \rfloor} f_0 \ge f_1 + 1$. *Then* $\log_d(N, m, p)$ *is multirate strictly nonblocking for* $0 \le m \le n-1$ *if and only if*

$$p \ge \max_{1 \le q \le Q} \left\{ \left\lfloor 2 g_m(q) + \frac{1}{d^m} \sum_{m+1}^{\lfloor \frac{n+m-1}{2} \rfloor} l_k(q) + \frac{1}{d^m} \sum_{m+1}^{\lceil \frac{n+m-1}{2} \rceil} l_k(q) \right\rfloor \right\} + 1.$$

*Proof.* The proof is analogous to the proof of Theorem 2.2. The assumption $f_{j-1} \le f_j$ for $j = 1, 2, \ldots, \lceil \frac{n+m-1}{2} \rceil$ is needed to guarantee

$$\frac{f_0}{f_k - q + 1} \cdot \frac{1}{d^k} > \frac{f_0}{f_{k+1} - q + 1} \cdot \frac{1}{d^{k+1}} \qquad \text{for} \quad 1 \le k \le m,$$

$$\frac{f_0}{f_k - q + 1} \cdot \frac{1}{d^m} > \frac{f_0}{f_{k+1} - q + 1} \cdot \frac{1}{d^m} \qquad \text{for} \quad 1 \le k \le \left\lfloor \frac{n+m-1}{2} \right\rfloor,$$

such that if an intersecting path intersects at several stages, the blocking effect is always greatest at the outmost stage, justifying our assigning it to that stage. ∎

**3. The continuous model.** We first quote a lemma proved by Melen and Turner [5].

LEMMA 3.1. $\lfloor \frac{a-w}{b-w+\epsilon} \rfloor = \lceil \frac{a-w}{b-w} \rceil - 1$ *if* $a \ge b$, *where* $\epsilon$ *is positive and tends to* 0.

THEOREM 3.2. *Consider the* $(b, B, \beta)$ *continuous model satisfying* $b + B \le 1$. *Then* $\log_d(N, 0, p)$ *is strictly nonblocking if*

$$p \ge \left\lceil \frac{d^{\lfloor \frac{n-1}{2} \rfloor} \beta - B}{1 - B} \right\rceil + \left\lceil \frac{d^{\lceil \frac{n-1}{2} \rceil} \beta - B}{1 - B} \right\rceil - 1,$$

*and the condition is necessary if* $b = 0$.

*Proof.* With an argument analogous to the proof of Theorem 2.1, we obtain the sufficient condition to route an $(i, o, w)$ new request to be

$$\begin{aligned}&\left\lfloor \frac{d^{\lfloor \frac{n-1}{2} \rfloor}\beta - w}{1-w+\epsilon} \right\rfloor + \left\lfloor \frac{d^{\lceil \frac{n-1}{2} \rceil}\beta - w}{1-w+\epsilon} \right\rfloor + 1 \\ &= \left\lceil \frac{d^{\lfloor \frac{n-1}{2} \rfloor}\beta - w}{1-w} \right\rceil + \left\lceil \frac{d^{\lceil \frac{n-1}{2} \rceil}\beta - w}{1-w} \right\rceil - 1,\end{aligned}$$

which is maximized at $w = B$.

To prove the necessity, note that for $b = 0$ we can always generate requests with suitable weights such that every saturated link carries a load of $1 - w - \epsilon$. □

THEOREM 3.3. *Consider the $(b, B, \beta)$ continuous model satisfying $b + B > 1$. Then $\log_d(N, 0, p)$ is strictly nonblocking if and only if*

$$p \geq \left\lfloor \frac{\beta}{b} \right\rfloor (d^{\lfloor \frac{n-1}{2} \rfloor} - 1) + \left\lfloor \frac{\beta}{b} \right\rfloor (d^{\lceil \frac{n-1}{2} \rceil} - 1) + 1.$$

*Proof.* Let the new request be $(i, o, w)$. Note that the channel graph between $i$ and $o$ is just a single path $l$. Similar to the proof of Theorem 2.1, $d^{\lfloor (n-1)/2 \rfloor} - 1$ inputs other than $i$ can generate $j$-intersecting paths for $j$ in $H_1$, and $d^{\lceil (n-1)/2 \rceil} - 1$ outputs can generate $j$-intersecting paths for $j$ in $H_2$. Since either an input or an output can generate at most $\lfloor \beta/b \rfloor$ requests, then

$$\left\lfloor \frac{\beta}{b} \right\rfloor (d^{\lfloor \frac{n-1}{2} \rfloor} - 1) + \left\lfloor \frac{\beta}{b} \right\rfloor (d^{\lceil \frac{n-1}{2} \rceil} - 1)$$

is an upper bound of the number of intersecting paths, and hence an upper bound of the number of blocked copies. Thus one extra copy suffices to carry the new request. Note that whether the extra copy carries any load from $i$ or $o$ is immaterial since the load cannot exceed $\beta - w$.

On the other hand, suppose $w = B$. Then, analogous to Theorem 2.1, the worst case described above can occur and the new request cannot be routed through any link already carrying a load $b$. Hence the sufficient condition is also necessary. □

Next we consider $m > 0$ case. Define

$$g_m(w) = \sum_{j=1}^{m} \frac{1}{d^j} \left\{ \left\lceil \frac{d^j\beta - w}{1-w} \right\rceil - \left\lceil \frac{d^{j-1}\beta - w}{1-w} \right\rceil \right\} \qquad \text{for} \quad 1 \leq m \leq n-1.$$

THEOREM 3.4. *Consider the $(b, B, \beta)$ continuous model satisfying $b + B \leq 1$. Then $\log_d(N, m, p)$ is strictly nonblocking if*

$$\begin{aligned}p \geq \Bigg\lfloor 2g_m(B) &+ \left\lceil \frac{d^{\lfloor \frac{n+m-1}{2} \rfloor}\beta - B - (\lceil \frac{d^m\beta - B}{1-B} \rceil - 1)(1-B)}{(1-B)} \right\rceil \frac{1}{d^m} \\ &+ \left\lceil \frac{d^{\lceil \frac{n+m-1}{2} \rceil}\beta - B - (\lceil \frac{d^m\beta - B}{1-B} \rceil - 1)(1-B)}{(1-B)} \right\rceil \frac{1}{d^m} \Bigg\rfloor - 1,\end{aligned}$$

*and the condition is necessary if $b = 0$.*

*Proof.* The proof is analogous to the proof of Theorem 2.2. □

THEOREM 3.5. *Consider the $(b, B, \beta)$ continuous model satisfying $b + B > 1$. Then $\log_d(N, m, p)$ is strictly nonblocking if and only if*

$$p \geq 2\left\lfloor \frac{m(d-1)\lfloor \beta/b \rfloor}{d} \right\rfloor + \left\lfloor \frac{(d^{\lfloor \frac{n+m-1}{2} \rfloor} - d^m)\lfloor \beta/b \rfloor}{d^m} \right\rfloor + \left\lfloor \frac{(d^{\lceil \frac{n+m-1}{2} \rceil} - d^m)\lfloor \beta/b \rfloor}{d^m} \right\rfloor + 1.$$

*Proof.* Each input can generate at most $\lfloor \beta/b \rfloor$ requests. Let each internal link carry at most one request. Then there are $d^j - d^{j-1}$ inputs generating $(d^j - d^{j-1})\lfloor \beta/b \rfloor$ requests to intersect a stage-$j$ link in the $(i, o)$ channel graph for $1 \leq j \leq m$. Since each such intersecting path blocks $1/d^j$ copies of $BY_d^{-1}(n, m)$, a total of

$$\left\lfloor \sum_{j=1}^{m} \frac{(d^j - d^{j-1})\lfloor \beta/b \rfloor}{d^j} \right\rfloor = \left\lfloor \frac{m(d-1)\lfloor \beta/b \rfloor}{d} \right\rfloor$$

copies is blocked. Similarly, the output side blocks the same number of copies. Finally, stage $m + 1$ to stage $n - m - 1$ consists of $d^m$ copies of $BY_d^{-1}(n - m, 0)$. We use an argument analogous to the proof of Theorem 2.2 to compute the number of copies blocked in these stages to be

$$\left\lfloor \frac{(d^{\lfloor \frac{n+m-1}{2} \rfloor} - d^m)\lfloor \beta/b \rfloor}{d^m} \right\rfloor + \left\lfloor \frac{(d^{\lceil \frac{n+m-1}{2} \rceil} - d^m)\lfloor \beta/b \rfloor}{d^m} \right\rfloor.$$

So one extra copy suffices to route the new request.

To prove the necessity, let $w = B$. Then the worst case discussed above can occur. ◻

## REFERENCES

[1] D. G. CANTOR, *On nonblocking switching networks*, Networks, 1 (1971), pp. 367–377.

[2] S.-P. CHUNG AND K. W. ROSS, *On nonblocking multirate interconnection networks*, SIAM J. Comput., 20 (1991), pp. 726–736.

[3] C. CLOS, *A study of non-blocking switching networks*, Bell System Tech. J., 32 (1953), pp. 406–424.

[4] C.-T. LEA, *Multi-*$\log_2 N$ *networks and their applications in high-speed electronic and photonic switching systems*, IEEE Trans. Commun., 38 (1990), pp. 1740–1749.

[5] R. MELEN AND J. S. TURNER, *Nonblocking multirate networks*, SIAM J. Comput., 18 (1989), pp. 301–313.

[6] D.-J. SHYY AND C.-T. LEA, *Log*$_2(N, m, p)$ *strictly nonblocking networks*, IEEE Trans. Commun., 39 (1991), pp. 1502–1510.

SIAM J. Comput. © 2005 Society for Industrial and Applied Mathematics
Vol. 34, No. 6, pp. 1279–1301

# SRT DIVISION ALGORITHMS AS DYNAMICAL SYSTEMS[*]

MARK MCCANN[†] AND NICHOLAS PIPPENGER[†]

**Abstract.** Sweeney–Robertson–Tocher (SRT) division, as it was discovered in the late 1950s, represented an important improvement in the speed of division algorithms for computers at the time. A variant of SRT division is still commonly implemented in computers today. Although some bounds on the performance of the original SRT division method were obtained, a great many questions remained unanswered. In this paper, the original version of SRT division is described as a dynamical system. This enables us to bring modern dynamical systems theory, a relatively new development in mathematics, to bear on an older problem. In doing so, we are able to show that SRT division is ergodic, and is even Bernoulli, for all real divisors and dividends. With the Bernoulli property, we are able to use entropy to prove that the natural extensions of SRT division are isomorphic by way of the Kolmogorov–Ornstein theorem. We demonstrate how our methods and results can be applied to a much larger class of division algorithms.

**Key words.** SRT division, ergodic, Bernoulli, dynamical systems, entropy

**AMS subject classifications.** 68W40, 37E05

**DOI.** 10.1137/S009753970444106X

## 1. Introduction.

Since the discovery of the first radix-2 Sweeney–Robertson–Tocher (SRT) division algorithm, the use of the term "SRT division" has expanded to include a wide variety of higher radix nonrestoring division algorithms that are loosely based on the original. For example, there is the infamous implementation of a radix-4 SRT division algorithm in the first release of the Pentium CPU that has become widely known as the "Pentium Bug." One major difference between this implementation of radix-4 SRT division and the original radix-2 SRT division is that the former produces a constant number of quotient bits per step, while the latter produces a variable number. Modern implementations of SRT division use carry-save adders to perform additions and subtractions in constant time. Earlier implementations, however, used carry-propagate adders with delays that grow with the word length. Therefore, the primary goal of the early investigators was to reduce the number of uses of the costly adder. In the late 1950s, Sweeney [3], Robertson [17], and Tocher [21] independently made the observation that whenever a partial remainder is in the range $(-\frac{1}{2}, \frac{1}{2})$, there will be one or more leading zeros that can be shifted through in a very short amount of time (usually one cycle) thereby reducing the use of the adder. Although the aforementioned have received most of the credit for the algorithm named after them, it can be argued that Nadler described an equivalent algorithm in a 1956 paper [13]. The description of higher radix SRT division, which is the basis for modern SRT division, is generally attributed to Atkins [1], but this is not the version of division that we will be concerned with in this paper.

[*]Received by the editors February 16, 2004; accepted for publication (in revised form) February 3, 2005; published electronically June 27, 2005. A preliminary version of this paper appears in [11].
http://www.siam.org/journals/sicomp/34-6/44106.html

[†]Department of Computer Science, Princeton University, 35 Olden Street, Princeton, NJ 08544 (mmccann@cs.princeton.edu, njp@cs.princeton.edu). The research of the first author was supported by an NSERC research grant. It was conducted while the author was at the University of British Columbia. The research of the second author was supported by an NSERC research grant and by a Canada Research Chair. It was conducted while the author was at the University of British Columbia.

Although what is considered to be "costly" for a division algorithm has changed, it is still interesting to study and important to understand the behavior of successive partial remainders on average for a given divisor. Surprisingly, some of the most basic questions that one might have concerning the behavior of partial remainders for even simple radix-2 SRT division have remained unanswered for over 40 years. The difficulty that early investigators experienced in answering such questions was mainly due to a lack of necessary mathematical tools and results. During that past 30 years, the field of "dynamical systems theory" or "ergodic theory" has come into existence in mathematics and has been greatly developed. In this paper we show how to apply some of what is now known in dynamical systems theory to the earliest version of SRT division. In doing so, we are able to prove several previously unknown properties for simple SRT division. The results are quite general and can be adapted to other division algorithms. We view the value of these results as lying in the establishment of a connection between a well-developed area of mathematics and digital division, rather than in any practical consequences for division algorithms. For the remainder of this paper, the term SRT division will refer to the original algorithm unless otherwise stated.

The SRT division algorithms analyzed by Freiman [5] and Shively [20] are the same, but the authors differ in what they take to be a step of the algorithm: Freiman defines a step to be the operations from one use of the adder to the next, while Shively defines it to be the operations from one normalizing shift (of a single place) to the next. The following definitions are consistent with Shively's:

1. $n$ represents the number of iterations performed in the algorithm.
2. $p_0$ is the dividend (or initial partial remainder) normalized so that $p_0 \in [\frac{1}{2}, 1)$.
3. $p_i \in (-1, 1)$, $i \in \mathbb{N}$, is the partial remainder after the $i$th step.
4. $D$ is the divisor normalized to $[\frac{1}{2}, 1)$.
5. $q_i \in \{-1, 0, 1\}$ $(i \in \{0, \ldots, n-1\})$ is the quotient digit generated by the $i$th step.
6. $Q_n = \sum_{i=0}^{n-1} \frac{q_i}{2^i}$ is the "rounded off" quotient generated after $n$ steps of the algorithm.

Given the above definitions, after $n$ steps of the division algorithm, we would like it to be true that

$$p_0 = DQ_n + \varepsilon(n),$$

where $\varepsilon(n)$ is a term that goes to zero as $n$ goes to infinity.

A recurrence relation for the SRT division algorithm can be stated as

$$(p_{i+1},\ q_i) = \begin{cases} (2p_i, & 0) & : & |p_i| < \frac{1}{2}, \\ (2(p_i - D), & 1) & : & |p_i| \geq \frac{1}{2} \text{ and } p_i \geq 0, \\ (2(p_i + D), & -1) & : & |p_i| \geq \frac{1}{2} \text{ and } p_i < 0. \end{cases}$$

By observing that

$$p_{i+1} = \begin{cases} 2(p_i - (0)D) & : & |p_i| < \frac{1}{2}, \\ 2(p_i - (1)D) & : & |p_i| \geq \frac{1}{2} \text{ and } p_i \geq 0, \\ 2(p_i - (-1)D) & : & |p_i| \geq \frac{1}{2} \text{ and } p_i < 0, \end{cases}$$

we can rewrite the definition of $p_{i+1}$ as

$$p_{i+1} = 2(p_i - q_i D)\,.$$

After $n$ steps have been completed, we have

$$p_n = 2^n p_0 - 2^n q_0 D - 2^{n-1} q_1 D - \cdots - 2^1 q_{n-1} D \,,$$

and then after dividing by $2^n$ and solving for $p_0$ we find that

$$\begin{aligned} p_0 &= \frac{p_n}{2^n} + \frac{q_0 D}{2^0} + \frac{q_1 D}{2^1} + \cdots + \frac{q_{n-1} D}{2^{n-1}} \\ &= D \sum_{i=0}^{n-1} \frac{q_i}{2^i} + \frac{p_n}{2^n} = D Q_n + \frac{p_n}{2^n} \,. \end{aligned}$$

Now let $\varepsilon(n) = p_n / 2^n$ and let $Q^* = \lim_{n\to\infty} Q_n$. Since $|p_n| < 1$, in the limit as $n$ goes to infinity,

$$p_0 = D Q^* \,.$$

The generated quotient bits ($-1$, 0, $+1$ valued) are not in a standard binary representation, but it is a simple matter to convert the answer back to standard binary without using any expensive operations. Table 1 shows an example of using the SRT division algorithm to divide 0.67 by 0.75. The steps that produce nonzero quotient bits have been shown. In this example, after six uses of the adder, the quotient ($0.89\overline{3}$) has been determined to four digits of precision.

Table 1
*SRT division where* $p_0 = 0.67$ *and* $D = 0.75$.

| | | | |
|---|---|---|---|
| $p_0 = 0.67$ | $= \phantom{-}0.67$ | | |
| $p_1 = 2(0.67 - D)$ | $= -0.16$ | $q_0 = 1$ | $Q_0 = 1$ |
| $p_4 = 2(2^2(-0.16) + D)$ | $= \phantom{-}0.22$ | $q_3 = -1$ | $Q_3 = 0.875$ |
| $p_7 = 2(2^2(0.22) - D)$ | $= \phantom{-}0.26$ | $q_6 = 1$ | $Q_6 = 0.890625$ |
| $p_9 = 2(2^1(0.26) - D)$ | $= -0.46$ | $q_8 = 1$ | $Q_8 = 0.89453125$ |
| $p_{11} = 2(2^1(-0.46) + D)$ | $= -0.34$ | $q_{10} = -1$ | $Q_{10} = 0.8935546875$ |
| $p_{13} = 2(2^1(-0.34) + D)$ | $= \phantom{-}0.14$ | $q_{12} = -1$ | $Q_{12} \doteq 0.8933105469$ |

Now, with this simple system of division in hand, we might want to ask certain questions about its performance. For example, we could ask, "How many bits of precision are generated per iteration of the algorithm on average?" To answer this question, we must look at the magnitude of $|Q^* - Q_n| = |p_n/2^n|$. The number of bits of precision on the $n$th step is then $n - \log_2 p_n$. In the worst case, $p_n$ is close to 1, and therefore we get at least one bit of precision per iteration of the algorithm, regardless of the values of $D$ or $p_0$. Of course, a designer of actual floating-point hardware probably wants to know the expected performance based on the expected values of $p_n$. To answer the many variants of this type of question, it is clear that we must know something about the distribution of partial remainders over time. The remainder of this paper is devoted to extending what is known about the answer to this type of question as it relates to SRT division and its variants.

**2. SRT division as a dynamical system.** The example in Table 1 makes it clear that keeping track of the signs of successive partial remainders is irrelevant in determining how many times the adder will be used for a particular calculation. For this reason, we need only consider the magnitudes of successive partial remainders. We now give a reformulation of SRT division that will allow us to look at division as a dynamical system.

DEFINITION 1 (SRT division transformation). *For $D \in [\frac{1}{2}, 1)$, we define the function $T_D : [0,1) \to [0,1)$ as*

$$T_D(x) = \begin{cases} 2x & : \quad 0 \le x < \frac{1}{2}, \\ 2(D - x) & : \quad \frac{1}{2} \le x < D, \\ 2(x - D) & : \quad D \le x < 1. \end{cases}$$

*This transformation of the unit interval represents the successive partial remainders that arise as SRT division is carried out by a divisor $D$ on a dividend $x$. $D$ is normalized to $[\frac{1}{2}, 1)$. The dividend $x$ is normalized to $[\frac{1}{2}, 1)$ initially, while each of the successive partial remainders $T_D^n(x)$ ($n \in \mathbb{N}$) subsequently ranges through $[0, 1)$.*

By using the characteristic function for a set $\Delta$ defined as

$$1_\Delta(x) = \begin{cases} 1 & : \quad x \in \Delta, \\ 0 & : \quad x \notin \Delta, \end{cases}$$

we can rewrite $T_D$ as

$$T_D(x) = 2x \cdot 1_{[0,\frac{1}{2})}(x) \, + \, 2(D - x) \cdot 1_{[\frac{1}{2},D)}(x) \, + \, 2(x - D) \cdot 1_{[D,1)}(x) \; . \tag{1}$$

If we plot (1) on the unit interval, we obtain a very useful visualization of our transformation. Figure 1 shows the plot of $T_{0.75}(x)$ combined with a plot of the successive partial remainders that arise while dividing 0.67 by 0.75. This is the same system that was presented earlier in Table 1. Notice that a vertical line in the interval $[\frac{1}{2}, D)$ corresponds to a subsequent flip in the sign of the next partial remainder.

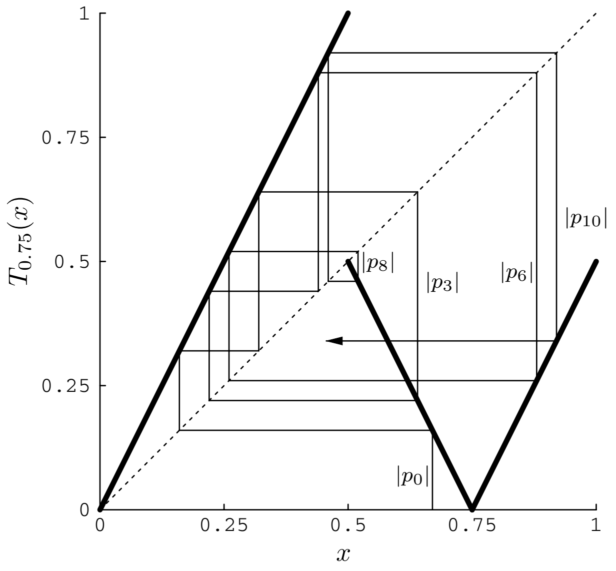


FIG. 1. *Graphic representation of partial remainder magnitudes for $D = 0.75$ and $p_0 = 0.67$.*

Figure 1 shows an example of following the trajectory of a single partial remainder for a particular divisor. In this figure, the heavy solid lines represent the transformation $T_{0.75}$, while the abscissa of the thin vertical lines represents successive partial remainder magnitudes. After 10 applications of the $T_{0.75}$, there is not any obvious regular pattern, although we expect to see one eventually since the quotient is rational in this case.[1] Of course, most numbers are not rational and we can deduce that for most numbers, the transformation will never exhibit a repeating pattern. In Figures 2 and 3, we see that a very small change in the value of the initial partial remainder quickly produces large differences in the observed behavior of the subsequent partial remainders. As we show in the appendix, our system is actually chaotic, and therefore we will gain little understanding by studying the trajectories of individual partial remainders. The logical next step is to study the behavior of distributions of points over the whole interval.

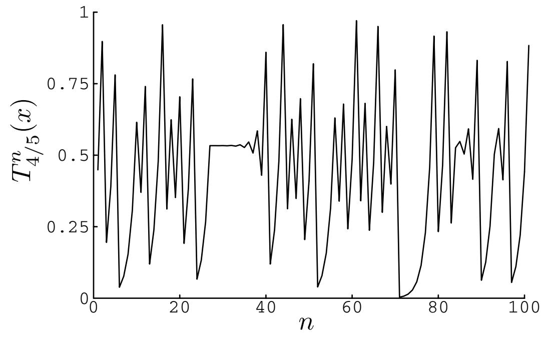


FIG. 2. *The result of applying $T_{4/5}$ to $x = \frac{\pi}{7}$ one hundred times.*

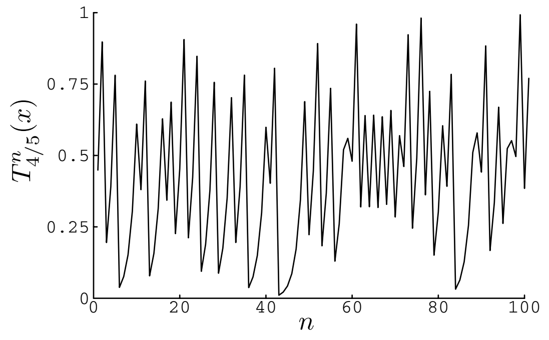


FIG. 3. *The result of applying $T_{4/5}$ to $x = \frac{\pi}{7} + 0.00001$ one hundred times.*

Understanding the behavior of ensembles of points under repeated transformation is in the realm of dynamical systems theory. For the remainder of this paper, we assume a certain amount of familiarity with the fundamentals of dynamical systems theory (or ergodic theory), which requires some basic understanding of measure theory. We will include a few helpful background material definitions as they are needed, but mostly we will provide references. A very good introduction to the study of chaotic systems is Lasota and Mackey's book [8]. For a more detailed introduction to ergodic theory (along with the necessary measure theory needed to understand this paper), Walters's book [22] and Petersen's book [16] are highly recommended.

[1]With redundant representations, rational numbers can have aperiodic representations, though we do not expect this to happen.

Definition 2 (probability space). *If $\mathcal{B}$ is a $\sigma$-algebra on subsets of a set $X$ and if $m$ is a measure on $\mathcal{B}$, where $m(X) = 1$, then the triple $(X, \mathcal{B}, m)$ is called a probability space.* (*See* [22, *pp.* 3–9] *and* [8, *pp.* 19–31] *for a good overview of basic measure theory and Lebesgue integration.*)

Definition 3 (Perron–Frobenius operator). *For a probability space $(X, \mathcal{B}, m)$,*[2] *the Perron–Frobenius operator $P : L^1 \to L^1$ associated with a nonsingular transformation $T : X \to X$ is defined by*

$$\int_B Pf(x)\,\mathrm{d}m = \int_{T^{-1}(B)} f(x)\,\mathrm{d}m \quad \textit{for } B \in \mathcal{B}\ .$$

For a piecewise monotonic $C^2$ transformation[3] $T$ with $n$ monotonic pieces, we can give an explicit formula for the Perron–Frobenius operator. Let $A = \{A_1, A_2, \dots, A_n\}$ be the partition of $X$ which separates $T$ into $n$ pieces. For $i \in \{1, \dots, n\}$, let $t_i(x)$ represent the natural extension of the $i$th $C^2$ function $T(x)|_{A_i}$. The Perron–Frobenius operator for $T$ is then

$$Pf(x) = \sum_{i=1}^{n} \left| \frac{\mathrm{d}}{\mathrm{d}x} t_i^{-1}(x) \right| f(t_i^{-1}(x)) \cdot 1_{t_i(A_i)}(x)\ .$$

In particular, for $T_D$ (as in (1)),

$$Pf(x) = \tfrac{1}{2} f(\tfrac{1}{2}x) \cdot 1_{[0,1)}(x) \;+\; \tfrac{1}{2} f(D - \tfrac{1}{2}x) \cdot 1_{(0,2D-1]}(x) \;+\; \tfrac{1}{2} f(D + \tfrac{1}{2}x) \cdot 1_{[0,2-2D)}(x). \tag{2}$$

With (2) we can show precisely what happens to an initial distribution of points (described by an integrable function) after they are repeatedly transformed under $T_D$. Figures 4 and 5 show what happens to two different initial distributions of points after five applications of the Perron–Frobenius operator associated with $T_{3/5}(x)$. By the fifth application, the distributions look remarkably similar. One might guess that they are both approaching the same final distribution. This situation is in marked contrast to chaotic behavior observed in Figures 2 and 3.

Definition 4 (stationary distribution). *Let $(X, \mathcal{B}, m)$ be a probability space, let $P$ be the Perron–Frobenius operator associated with a nonsingular transformation $T : X \to X$, and let $L^1$ denote the $L^1$ space of $(X, \mathcal{B}, m)$. If $f \in L^1$ is such that $Pf = f$ a.e.,*[4] *then $f$ is called a stationary distribution of $T$.*

A practical use of the Perron–Frobenius operator is in deriving and verifying the equations of stationary distributions for given divisors. As an example of this, we verify the correctness of a previously known stationary distribution for $D \in [\frac{3}{4}, 1)$. An exact equation was first given by Freiman [5] and is restated by Shively [20] as

$$f(x) = \frac{1}{D} 1_{[0,2D-1)}(x) + \frac{1}{2D} 1_{[2D-1,1)}(x)\,. \tag{3}$$

This relation can be verified by applying the Perron–Frobenius operator as given in (2) to (3). Such exact equations are not known in general for all $D \in [\frac{1}{2}, \frac{3}{4})$. This issue is discussed further in the conclusion.

[2]For a probability space $(X, \mathcal{B}, m)$, the $L^1$ space of $(X, \mathcal{B}, m)$ is the set of $f : X \to \mathbb{R}$ satisfying $\int_X |f(x)|\,\mathrm{d}m < \infty$.

[3]$C^2$ denotes the set of all functions with two continuous derivatives.

[4]a.e. indicates that the given relation holds *almost everywhere*, that is, everywhere except possibly on a set of measure zero.

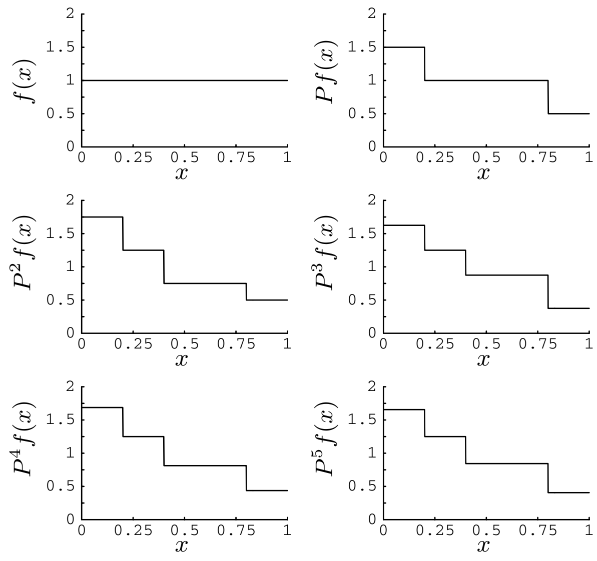


FIG. 4. *The result of applying the Perron–Frobenius operator $P$ associated with $T_{3/5}$ to $f(x) = 1$ six times.*

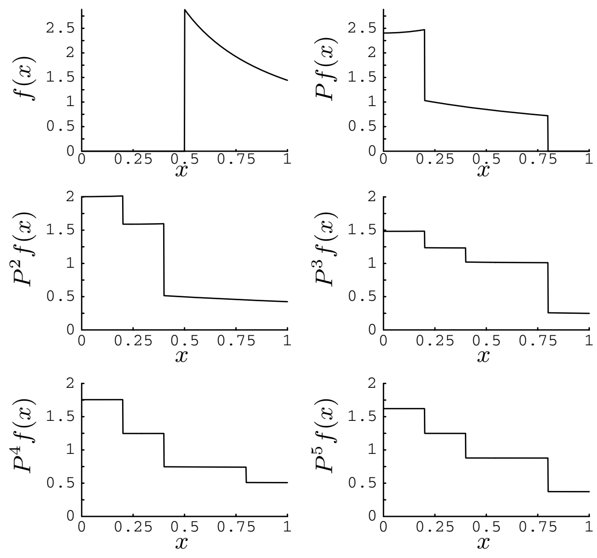


FIG. 5. *The result of applying the Perron–Frobenius operator $P$ associated with $T_{3/5}$ to $f(x) = (1/x \log_e 2) \cdot 1_{[\frac{1}{2},1)}(x)$ six times.*

In the case of variable quotient-bits-per-cycle algorithms such as the original SRT division, one of the primary uses of a formula for the distribution of partial remainders is for calculating the *shift average* for a given divisor. (Note that higher shift averages are desirable.) The shift average is the average number of uses of the shift register (single shift or multiplication by two) between uses of the adder. Under the assumption that a register shift is a much faster operation than using the adder, the shift average gives a useful characterization of the expected performance of our algorithm for a given divisor. With (3), we know the fraction of bits which require the use of the adder. To calculate the average number of zero bits generated between nonzero bits (bits requiring use of the adder), we take the reciprocal of the fraction of bits which do not require the adder. For $D \in [\frac{3}{4}, 1)$, the shift average function is

$$s(D) = \left(1 - \frac{1}{2D}\right)^{-1} = \frac{2D}{2D-1}\,. \tag{4}$$

Unfortunately, since we have not proven that the stationary distributions from SRT division are unique, we have no way of knowing whether or not a shift average calculation in (4) is correct for all initial probability distributions. To prove that all stationary distributions are unique, we need to show that $T_D$ is ergodic for all $D \in [\frac{1}{2}, 1)$.

Definition 5 (ergodic; see [8]). *Let $(X, \mathcal{B}, m)$ be a probability space and let a nonsingular transformation $T : X \to X$ be given. Then $T$ is ergodic if for every set $B \in \mathcal{B}$ such that $T^{-1}(B) = B$, either $m(B) = 0$ or $m(X \setminus B) = 0$.*

Freiman [5] shows that $T_D$ is ergodic for rational $D$, but we extend this result to real $D$. In the next section we show that all $T_D$ are Bernoulli and it is known that having the Bernoulli property implies ergodicity.

## 3. Bernoulli property.

Our central result, which we present in this section, is that the class of transformations of the interval that characterizes SRT division for all real divisors $D$ has the property that each transformation $T_D$ is Bernoulli. Although the basic concept of a Bernoulli shift (the things to which transformations having the Bernoulli property are isomorphic) is not difficult, a complete definition requires enough auxiliary concepts from measure theory (concepts not used anywhere else in this paper) that we refer the interested reader to [15, 16, 19, 22]. Neither an understanding of Bernoulli shifts nor a formal definition of what it means to be Bernoulli is required to follow the proofs in this section. Having said this, we should mention informally the connection between Bernoulli shifts and transformations having the Bernoulli property.

The transformation $T_D$ is a noninvertible endomorphism of the unit interval. This means that from a given partial remainder we can predict all future partial remainders, but we cannot uniquely predict past partial remainders. There is a natural way (called the natural extension) to make our transformation invertible (an automorphism) on a larger space. Specifically, each noninvertible transformation $T_D$ having the Bernoulli property has an extension to an automorphic transformation, isomorphic to a two-sided Bernoulli shift [16, pp. 13, 276]. From the way that entropy for a transformation is defined, the entropy for an automorphic Bernoulli transformation associated with a noninvertible Bernoulli transformation is the same as the entropy for the noninvertible Bernoulli transformation. By proving that all transformations $T_D$ are Bernoulli, and by proving that the entropy of each $T_D$ is the same, we will be able to conclude that the natural extensions of SRT division algorithms are isomorphic to each other for all divisors.

Definition 6 (expanding; see Bowen [2]). *We will say that a transformation $T$ on an interval is expanding if it has the property that $\sup_{n>0} \mu(T^n U) = 1$ for all open intervals $U$ with $\mu(U) > 0$, where $\mu$ is any normalized measure that is absolutely continuous with respect to Lebesgue measure.*

Definition 7 (straddle). *Let $U$ be an interval of reals (either open, closed, or half open) and let $p \in \mathbb{R}^+$. If $p \in U^\circ$,*[5] *then we say that $U$ straddles $p$.*

Theorem 8. *The SRT division transformation is expanding for all real divisors.*

*Proof.* Let $(X, \mathcal{B}, m)$ be a probability space, where $X = [0,1)$, $\mathcal{B}$ is the Borel $\sigma$-algebra on $X$, and $m$ is the Lebesgue measure on $\mathcal{B}$.[6] Let $T_D : X \to X$ be the SRT division transformation for a given normalized divisor $D$ as defined in (1).

Let us define an infinite sequence of intervals $\mathcal{U} = \{U_i\}_{i\in\mathbb{N}}$ as

$$U_1 = U \quad \text{and}$$
$$U_{i+1} = \begin{cases} T_D(U_i) & : \quad U_i^\circ \subseteq [0, \frac{1}{2}) \qquad \text{or } U_i^\circ \subseteq [\frac{1}{2}, 1), \\ T_D(U_i \cap [0, \frac{1}{2})) & : \quad U_i^\circ \not\subseteq [0, \frac{1}{2}) \text{ and } U_i^\circ \not\subseteq [\frac{1}{2}, 1) \text{ and} \\ & \qquad\qquad m(U_i \cap [0, \frac{1}{2})) \geq m(U_i \cap [\frac{1}{2}, 1)), \\ T_D(U_i \cap [\frac{1}{2}, 1)) & : \quad U_i^\circ \not\subseteq [0, \frac{1}{2}) \text{ and } U_i^\circ \not\subseteq [\frac{1}{2}, 1) \text{ and} \\ & \qquad\qquad m(U_i \cap [0, \frac{1}{2})) < m(U_i \cap [\frac{1}{2}, 1)). \end{cases}$$

Property 1. *For all $U_i$ such that $\frac{1}{2} \notin U_i^\circ$ and $D \notin U_i^\circ$, $m(U_{i+1}) = 2m(U_i)$.*

*Proof.* If a $U_i^\circ$ is a subset of either $[0, \frac{1}{2})$, $[\frac{1}{2}, D)$, or $[D, 1)$, then we are in the first case of the $\mathcal{U}$ definition and we apply $T_D$ directly. Since each of the three cases of the $T_D$ expands an interval by a factor of two, it is clear that $m(T_D(U_i)) = m(U_{i+1}) = 2m(U_i)$.

Property 2. *For all $U_i$ where $D \notin U_i$, $m(U_{i+1}) \geq m(U_i)$.*

*Proof.* Assume that $D \notin U_i$. If $\frac{1}{2} \notin U_i$, then according to Property 1, $U_{i+1}$ doubles. Otherwise, $\frac{1}{2} \in U_i$, and therefore, to find $U_{i+1}$, we must consider the second and third cases of the $\mathcal{U}$ sequence. In the worst case, $m(U_i \cap [0, \frac{1}{2})) = m(U_i \cap [\frac{1}{2}, D))$, and regardless of which half we choose, $m(U_i \cap [0, \frac{1}{2})) = m(U_i \cap [\frac{1}{2}, D)) = \frac{1}{2}m(U_i)$. By applying $T_D$ to this truncated interval, we double what we halved so that $m(U_i) = m(U_{i+1})$.

By way of contradiction, let us assume that there exists an initial $U$ such that the sequence $\mathcal{U}$ never expands to fill $X$. Such a sequence can never include the point $D$; if it did, there would be a small interval about $D$ that would be mapped to $[0, \varepsilon)^\circ$, and this interval would quickly expand to fill the whole interval. We can show that the following property will hold.

Property 3. *There exists $N$ such that for all $i \geq N$,*

(a) $m(U_i \cap [0, \frac{1}{2})), m(U_i \cap [\frac{1}{2}, 1)) > 0$ *(in other words, all subsequent intervals must straddle $\frac{1}{2}$), and*

(b) $m(U_i \cap [0, \frac{1}{2})) < m(U_i \cap [\frac{1}{2}, 1))$ *(in other words, all subsequent $U_i$ must be such that the right half of $U_i$ is not discarded by the definition of $\mathcal{U}$).*

*Proof of Property* 3(a). Property 1 says that the only way not to double is to straddle $\frac{1}{2}$. Therefore, at a minimum, it must be the case that $\frac{1}{2}$ is eventually included every time or else the interval will double a sufficient number of times to include $D$, which would be a contradiction.

*Proof of Property* 3(b). If $m(U_i \cap [0, \frac{1}{2})) \geq m(U_i \cap [\frac{1}{2}, 1))$, then we have $U_i = (\frac{1}{2} - \varepsilon, \frac{1}{2} + \varepsilon')$, where $\varepsilon > \varepsilon'$. Now $U_{i+1} = T_D(U_i) = T_D(\frac{1}{2} - \varepsilon, \frac{1}{2}) = (1 - 2\varepsilon, 1)$. But,

[5]The symbol $\circ$ as the exponent of an interval denotes an open version of the interval.

[6]For an interval $[a, b]$, the Lebesgue measure is defined as $m([a, b]) = b - a$.

since $D$ is not in $U_{i+1}$, $\frac{1}{2}$ cannot be in $U_{i+1}$ and Property 3(a) fails, resulting in a contradiction.

By Property 3(a), we will eventually be in a situation where $U_i = (\frac{1}{2} - \varepsilon', \frac{1}{2} + \varepsilon)$, $\varepsilon' < \varepsilon$, and Property 3(a) will hold for every subsequent interval. So then

$$U_{i+1} = T_D(\tfrac{1}{2} - \varepsilon', \tfrac{1}{2} + \varepsilon) = T_D[\tfrac{1}{2}, \tfrac{1}{2} + \varepsilon) = (2D - 1 - 2\varepsilon, 2D - 1]$$

by Property 3(b). But again by Property 3(a),

$$U_{i+2} = T_D(2D - 1 - 2\varepsilon, 2D - 1] = T_D[\tfrac{1}{2}, 2D - 1] = [2 - 2D, 2D - 1]\,.$$

It is now clear that $\frac{1}{2}$ is at the midpoint of $U_{i+2}$ and that we must now pick the left half of the interval, which contradicts Property 3(b). Therefore, $D$ will eventually be included in an interval and the sequence will expand to fill all of $X$. ∎

We can now prove that the SRT division process is weak-mixing, and therefore Bernoulli, by two theorems of Bowen [2].

THEOREM 9 (see Bowen [2]). *Let $T$ be a piecewise $C^2$ map of $[0,1]$, $\mu$ be a smooth $T$-invariant probability measure, and $\lambda = \inf_{0\le x\le 1} |f'(x)| > 1$. If the dynamical system $(T,\mu)$ is weak-mixing, then the natural extension of $(T,\mu)$ is Bernoulli.*

We mention here that the *natural extension* of $(T,\mu)$ is the associated automorphic transformation that we alluded to at the beginning of this section. See Petersen [16, p. 13] for an exact definition.

THEOREM 10 (see Bowen [2]). *With $T$ and $\mu$ as in Theorem 9, $(T,\mu)$ will be weak-mixing if $T$ is expanding.*

THEOREM 11 (see Lasota and Yorke [9]). *Let $(X, \mathcal{B}, m)$ be a probability space and let $T : X \to X$ be a piecewise $C^2$ function such that $\inf |T'| > 1$. If $P$ is the Perron–Frobenius operator associated with $T$, then for any $f \in L^1$, the sequence $(\frac{1}{n}\sum_{k=0}^{n-1} P^k f)_{n=1}^{\infty}$ is convergent in norm to a function $f^* \in L_1$. The limit function $f^*$ has the property that $Pf^* = f^*$ and, consequently, the measure $\mathrm{d}\mu^* = f^*\,\mathrm{d}m$ is invariant under $T$.*

Having established that $T_D$ is expanding, we now use the above three theorems to prove the central result of this paper.

THEOREM 12. *$T_D$ is Bernoulli.*

*Proof.* From the definition of $T_D$, we see that $T_D$ is $C^2$ and that

$$\inf_{0\le x\le 1} \left|{T_D}'(x)\right| = 2 > 1$$

since $\left|{T_D}'(x)\right| = 2$ for all $x$, for which the derivative is defined. Since

$$\inf_{0\le x\le 1} \left|{T_D}'(x)\right| > 1,$$

by Theorem 11 there exists at least one $\mu$ such that $\mu$ is a smooth $T_D$-invariant probability measure. By Theorem 8, we see that Theorem 10 holds. Hence, $(T_D,\mu)$ is weak-mixing, and by Theorem 9 $(T_D,\mu)$ is Bernoulli. ∎

That all $T_D$ are Bernoulli is a very useful property because we can use entropy as a complete invariant to show isomorphism among the two-sided Bernoulli shifts associated with $T_D$ that have the same entropy. This comes from the contribution of Ornstein to the Kolmogorov–Ornstein theorem.

THEOREM 13 (see Kolmogorov [6, 7] and Ornstein [14]). *Two Bernoulli shifts are isomorphic if and only if they have the same entropy.*

Before we calculate the entropy, we review the definition of entropy for a transformation along with some supporting definitions that follow the development presented by Walters [22, pp. 75–87].

DEFINITION 14 (partition). *A partition of $(X, \mathcal{B}, m)$ is a disjoint collection of nonempty elements of $\mathcal{B}$ whose union is $X$.*

DEFINITION 15 (join). *Let $\mathcal{P}$ and $\mathcal{Q}$ be finite partitions of $(X, \mathcal{B}, m)$. Then $\mathcal{P} \vee \mathcal{Q} = \{P \cap Q : P \in \mathcal{P} \text{ and } Q \in \mathcal{Q}\} \setminus \{\emptyset\}$ is called the join of $\mathcal{P}$ and $\mathcal{Q}$. Note that $\mathcal{P} \vee \mathcal{Q}$ is also a finite partition of $(X, \mathcal{B}, m)$.*

DEFINITION 16 (entropy of a partition). *Let $(X, \mathcal{B}, m)$ be a probability space and let $\mathcal{P} = \{P_1, \ldots, P_k\}$ be a finite partition of $(X, \mathcal{B}, m)$. The entropy of the partition is defined as*

$$H(\mathcal{P}) = -\sum_{i=1}^{k} m(P_i) \log m(P_i) \, .$$

DEFINITION 17 (entropy of a transformation with respect to a partition). *Suppose $T : X \to X$ is a measure-preserving transformation of the probability space $(X, \mathcal{B}, m)$. If $\mathcal{P}$ is a finite partition of $(X, \mathcal{B}, m)$, then*

$$h(T, \mathcal{P}) = \lim_{n \to \infty} \frac{1}{n} H\left( \bigvee_{i=0}^{n-1} T^{-i}\mathcal{P} \right)$$

*is called the entropy of $T$ with respect to partition $\mathcal{P}$.*

DEFINITION 18 (entropy of a transformation). *Let $T : X \to X$ be a measure-preserving transformation of the probability space $(X, \mathcal{B}, m)$ and suppose $h(T) = \sup h(T, \mathcal{P})$, where the supremum is taken over all finite partitions $\mathcal{P}$ of $(X, \mathcal{B}, m)$. Then $h(T)$ is called the entropy of $T$.*

In general it can be very difficult to calculate the entropy for a class of transformations directly from the definition of entropy. Even with the many standard formulas that have been derived for calculating entropy, a great number of systems found in practice are not covered. Simple SRT division is one such dynamical system for which it is not easy to calculate the entropy from results found in standard textbooks on ergodic theory. Fortunately, a result by Ledrappier does allow us to calculate the entropy for simple SRT division.

The following definitions and theorems involving C-maps and PC-maps are taken from a paper of Ledrappier [10] and have been streamlined for our argument.

DEFINITION 19 (C-map; see Ledrappier [10]). *A real function $f$ defined on an interval $[a, b]$ is said to be a* C*-map if $f$ is continuously differentiable and its derivative $f'$ has the following properties:*

(a) *$f'$ satisfies a Hölder condition*[7] *of order $\varepsilon > 0$.*

(b) *There are only a finite number of points $x \in [a, b]$ where $f'(x) = 0$. We denote them by $a \le a_1 < a_2 \cdots < a_n \le b$ with $f'(a_i) = 0$ for $0 < i \le n$.*

(c) *There exist positive numbers $k_i^-$ ($k_i^+$) such that*

$$\left| \log \frac{|f'(x)|}{|x - a|^{k_i^{-(+)}}} \right|$$

*is bounded in a left (right) neighborhood of $a_i$.*

[7]A function $f(x)$ defined on an interval $[a, b]$ satisfies a Hölder condition of order $\varepsilon \in \mathbb{R}^+$ if there exists $c \in \mathbb{R}^+$ such that for any two points $p_1, p_2 \in [a, b]$, $|f(p_1) - f(p_2)| \le c\,|p_1 - p_2|^{\varepsilon}$.

Definition 20 (PC-map; see Ledrappier [10]). *A map $f : [0,1) \to [0,1)$ is called a* PC*-map if there exists a finite partition $0 < b_1 < b_2 \cdots < b_m < 1$ such that $f$ is a* C*-map from $[b_j, b_{j+1}]$ into $[0,1)$ for any $j$.*

Theorem 21 (see Ledrappier [10]). *Let $f$ be a* PC*-map. If $\mu$ is an a.c.i.m. (absolutely continuous invariant measure), then*

$$h(f) = \int \log |f'| \, \mathrm{d}\mu. \tag{5}$$

This formula was shown by Rohlin [18] to hold for a smaller class of transformations, which does not include the $T_D$ associated with SRT division.

Theorem 22. *The entropy $h(T_D)$ of $T_D$ for $D \in [\frac{1}{2}, 1)$ is equal to $\int \log \left|T_D{}'\right| \, \mathrm{d}\mu = \log 2$.*

*Proof.* By the definition of a PC-map, $T_D$ is a PC-map if each of the three functions $T_D|_{[0,\frac{1}{2})}$, $T_D|_{[\frac{1}{2},D)}$, and $T_D|_{[D,1)}$ is a C-map.

Trivially, each $T_D$ restricted to any of the three domains $[0, \frac{1}{2})$, $[\frac{1}{2}, D)$, and $[D, 1)$ satisfies a Hölder condition of order $\varepsilon = 1$ because each piece of $T_D$ is just a line of slope two. Thus condition (a) of Definition 19 is satisfied. Conditions (b) and (c) are satisfied because there are no points for which the derivative is equal to zero within a given line segment. Thus each of the three segments of $T_D$ are C-maps and, by Definition 20, $T_D$ is a PC-map.

Now, since each $T_D$ is Bernoulli, there exists a unique a.c.i.m. (call it $\mu$) for each $T_D$. By Theorem 21, we can use (5) to calculate the entropy:

$$h(T_D) = \int \log \left|T_D{}'\right| \mathrm{d}\mu = \log 2 \int \mathrm{d}\mu = \log 2. \qquad \square$$

With the proof of Theorem 22 we have established isomorphism among the automorphic transformations (or natural extensions) associated with simple SRT division transformations by an application of the Kolmogorov–Ornstein theorem. The key to obtaining this result was being able to show that $T_D$ has Bowen's expanding property. In the following section, we extend these results to a more general type of SRT division.

## 4. Multithreshold SRT division.

A simple optimization to the original SRT division algorithm, at least with the historical concern of avoiding additions and subtractions in mind, is the inclusion of additional divisors to increase the shift average. In this section, we prove that all such division algorithms with reasonable assumptions on the separation of the divisor multiples have the expanding property. It will be useful to precisely define a class of "multithreshold" SRT transformations.

Definition 23. *Let $\boldsymbol{\alpha} \in \mathbb{R}^n$ be such that*

(a) $0 < \alpha_1 < \alpha_2 < \cdots < \alpha_n$*, and*

(b) *for all $x, D \in [\frac{1}{2}, 1)$, there exists $i \in \{1, \ldots, n\}$ such that $|\alpha_i D - x| < \frac{1}{2}$.*

*We define $\mathfrak{A}_n$ to be the set of all $\boldsymbol{\alpha} \in \mathbb{R}^n$ satisfying conditions* (a) *and* (b)*. Also, $\mathfrak{A} = \bigcup_{n \in \mathbb{N}} \mathfrak{A}_n$.*

Definition 24 (peaks and valleys). *Given an $\boldsymbol{\alpha} \in \mathfrak{A}_{n \geq 2}$, the point of intersection between two lines $f(x) = 2(x - \alpha_i D)$ and $g(x) = 2(\alpha_{i+1} D - x)$ will be called a peak and is denoted by $\boldsymbol{\psi_i} = (\frac{1}{2} D(\alpha_{i+1} + \alpha_i), D(\alpha_{i+1} - \alpha_i))$. For convenience, we will refer to the abscissa as $\psi_i^x = \frac{1}{2} D(\alpha_{i+1} + \alpha_i)$ and to the ordinate as $\psi_i^y = D(\alpha_{i+1} - \alpha_i)$. The point of intersection of the two lines $f(x) = 2(\alpha_i D - x)$ and $g(x) = 2(x - \alpha_i D)$ is $(\alpha_i D, 0)$ and will be called a valley.*

Definition 25. *For a $D \in [\frac{1}{2}, 1)$ and an $\boldsymbol{\alpha} \in \mathfrak{A}$, define the transformation $T_{D,\boldsymbol{\alpha}}(x) : [0,1) \to [0,1)$. For $\boldsymbol{\alpha} \in \mathfrak{A}_1$, we get the familiar transformation*

$$T_{D,\boldsymbol{\alpha}}(x) = \begin{cases} 2x & : \quad 0 \le x < \frac{1}{2}, \\ |2(D-x)| & : \quad \frac{1}{2} \le x < 1\,. \end{cases}$$

*For $\boldsymbol{\alpha} \in \mathfrak{A}_2$,*

$$T_{D,\boldsymbol{\alpha}}(x) = \begin{cases} 2x & : \quad 0 \le x < \frac{1}{2}, \\ |2(\alpha_1 D - x)| & : \quad \frac{1}{2} \le x < \psi_1^x, \\ |2(\alpha_2 D - x)| & : \quad \frac{1}{2} \le x \text{ and } \psi_1^x \le x < 1\,. \end{cases}$$

*For $\boldsymbol{\alpha} \in \mathfrak{A}_{n\ge 3}$,*

$$T_{D,\boldsymbol{\alpha}}(x) = \begin{cases} 2x & : \quad 0 \le x < \frac{1}{2}, \\ |2(\alpha_1 D - x)| & : \quad \frac{1}{2} \le x < \psi_1^x, \\ |2(\alpha_i D - x)| & : \quad \frac{1}{2} \le x \text{ and } \psi_i^x \le x < \psi_{i+1}^x, \\ |2(\alpha_n D - x)| & : \quad \frac{1}{2} \le x \text{ and } \psi_{n-1}^x \le x < 1\,. \end{cases}$$

Definition 26. *Define $\mathfrak{M}_n = \{T_{D,\boldsymbol{\alpha}} : D \in (\frac{1}{2}, 1], \boldsymbol{\alpha} \in \mathfrak{A}_n\}$ and define $\mathfrak{M} = \bigcup_{n\in\mathbb{N}} \mathfrak{M}_n$. We call $\mathfrak{M}_n$ the set of all $n$-threshold SRT division transformations and call $\mathfrak{M}$ the set of multithreshold SRT division transformations.*

Table 2 shows an example of dividing 0.67 by 0.75 using multithreshold SRT division with $\boldsymbol{\alpha} = (0.75, 1, 1.25)$. This is the same example calculation as in Table 1, but here the dividend has been computed to twice as many digits of precision with the same effective number of uses of the adders. We say "effective" because in multithreshold SRT division, there are several adders working in parallel. In a real implementation of multithreshold SRT division, the values for $\boldsymbol{\alpha}$ must be carefully chosen so that not too much overhead is required to select a good partial remainder. There is also a tradeoff between the amount of overhead in choosing a good partial remainder and the precision to which a good partial remainder is selected.

Table 2
*An example of multithreshold SRT division.*

| | | |
|---|---|---|
| $p_0 = 0.67 = 0.67$ | | |
| $p_1 = 2(0.67 - \alpha_2 D) = -0.16$ | $q_0 = \alpha_2$ | $Q_0 = 1$ |
| $p_4 = 2(2^2(-0.16) + \alpha_1 D) = -0.155$ | $q_3 = -\alpha_1$ | $Q_3 = 0.90625$ |
| $p_7 = 2(2^2(-0.155) + \alpha_1 D) = -0.115$ | $q_6 = -\alpha_1$ | $Q_6 = 0.89453125$ |
| $p_{11} = 2(2^3(-0.115) + \alpha_3 D) = 0.035$ | $q_{10} = -\alpha_3$ | $Q_{10} \doteq 0.8933105469$ |
| $p_{16} = 2(2^4(0.035) - \alpha_1 D) = -0.005$ | $q_{15} = \alpha_1$ | $Q_{15} \doteq 0.8933334351$ |
| $p_{24} = 2(2^7(0.005) + \alpha_1 D) = -0.155$ | $q_{23} = -\alpha_1$ | $Q_{23} \doteq 0.8933333456$ |

Condition (b) in Definition 23 guarantees that the division algorithm generates a new quotient bit at every step. Although the condition makes sense intuitively, it is not immediately obvious just by inspection if an $\boldsymbol{\alpha}$ satisfies the condition. Lemma 28 below provides an easier way to check.

Lemma 27. *If $\boldsymbol{\alpha} = (\alpha_1)$, then condition* (b) *of Definition* 23 *is satisfied if and only if $\alpha_1 = 1$.*

*Proof.* If $\alpha_1 = 1$, then $\max_{D,x\in[1/2,1)} |\alpha_1 D - x| < \frac{1}{2}$. Now consider the cases when $\alpha_1 \ne 1$ and $\varepsilon \in \mathbb{R}^+$. If $\alpha_1 = 1 + \varepsilon$, then when $D = \frac{1}{1+\varepsilon}$ and $x = \frac{1}{2}$, $|\alpha_1 D - x| =$

$1 - \frac{1}{2} = \frac{1}{2} \not< \frac{1}{2}$. On the other hand, if $\alpha_1 = 1 - \varepsilon$, then when $D = \frac{1}{2}$ and $x = 1 - \frac{\varepsilon}{2}$, $|\alpha_1 D - x| = 1 - \frac{\varepsilon}{2} - (1-\varepsilon)\frac{1}{2} = \frac{1}{2} \not< \frac{1}{2}$. ◻

Lemma 28. *An $\boldsymbol{\alpha} \in \mathfrak{A}_n$ that satisfies condition* (a) *of Definition* 23 *also satisfies condition* (b) *if and only if for some $i, j \in \{1, \ldots, n\}$ (possibly $i = j$) either*

(i) $\alpha_i \in (0, \frac{1}{2}]$ *and* $\alpha_j \in [1, 1 + \alpha_i]$, *or*

(ii) $\alpha_i \in [\frac{1}{2}, 1]$ *and* $\alpha_j \in [1, 3\alpha_i]$.

*Proof* (sketch). Lemma 27 has shown that a single component $\alpha$ of $\boldsymbol{\alpha}$ with $\alpha = 1$ is sufficient to ensure that the range of $f(x) = 2\,|\alpha D - x|$ is equal to $[0, 1)$ as $x$ and $D$ range over $[\frac{1}{2}, 1)$. It is easy to see, based on the proof of Lemma 27, that if there does not exist $i \in \{1, \ldots, n\}$ such that $\alpha_i = 1$, then there must exist $i, j \in \{1, \ldots, n\}$ $(i < j)$, where $\alpha_i < 1$ and $\alpha_j > 1$.

Let us assume that $i$ is the largest value, where $\alpha_i < 1$, and let us assume that $j$ is the smallest value, where $\alpha_j > 1$ (therefore $j = i + 1$). We make these assumptions because no other scalars of $D$ will have an influence on whether or not condition (b) is satisfied. Consider the case when $\alpha_i \in (0, \frac{1}{2}]$. In this case, when $D$ is close enough to 1, some of the line $f(x) = 2(x - \alpha_i D)$ appears in the region (denoted $R$), when $\frac{1}{2} \le x < 1$, $0 \le T_{\boldsymbol{\alpha}}(x) < 1$. When a portion of the line $f(x)$ appears in region $R$, we must put restrictions on $\alpha_j$ in terms of $\alpha_i$ so that the peak $\boldsymbol{\psi}_1$ is always in $R$. $\psi_i^y$ is greatest when $D = 1$. We find the maximum allowable value of $\alpha_j$ by setting $D = 1$ and solving $\psi_i^y = 1$ for $\alpha_j$:

$$\psi_i^y = 1 \quad \Rightarrow \quad D(\alpha_j - \alpha_i) = 1 \quad \Rightarrow \quad \alpha_j = \alpha_i + 1\,.$$

Therefore, if $\alpha_i \in (0, \frac{1}{2}]$, then $\alpha_j \in [1, 1 + \alpha_i]$.

In the case when $\alpha_i \in [\frac{1}{2}, 1]$, for large enough values of $D$, the line $f(x) = 2(x - D\alpha_i)$ crosses the line $x = 1$ in the range $[0, 1)$. Because of this, we must loosen the restriction that $\alpha_j \in [1, 1 + \alpha_j]$. It is straightforward to calculate that $f(x)$ begins to cross the line $x = 1$ in the range $[0, 1)$ when $D = \frac{1}{2\alpha_i}$. By solving $\psi_i^y = 1$ for $\alpha_j$ when $D = \frac{1}{2\alpha_i}$ we can ensure that as $D$ becomes smaller, the peak $\boldsymbol{\psi_i}$ will always be in region $R$:

$$\psi_i^y = 1 \quad \Rightarrow \quad D(\alpha_j - \alpha_i) = 1 \quad \Rightarrow \quad \frac{1}{2\alpha_i}(\alpha_j - \alpha_i) = 1 \quad \Rightarrow \quad \alpha_j = 3\alpha_i\,.$$

Therefore, if $\alpha_i \in [\frac{1}{2}, 1]$, then $\alpha_j \in [1, 3\alpha_i]$. ◻

Definition 29 (separation). *For $\boldsymbol{\alpha} \in \mathfrak{A}_n$, we define the separation in $\boldsymbol{\alpha}$ as*

$$sep(\boldsymbol{\alpha}) = \max_{i \in \{1, \ldots, n-1\}} \frac{\alpha_{i+1}}{\alpha_i}\,.$$

*Limiting the separation is a convenient way to restrict the subset of $\mathfrak{A}$ being considered. If $sep(\boldsymbol{\alpha}) = r$, we say that "the divisor multiples in $\boldsymbol{\alpha}$ are separated by at most a factor of $r$."*

We are now ready to show that all multithreshold SRT division transformations are Bernoulli, given a necessary restriction on the multiples of the divisor. As in the case for a single divisor, it will be useful to define a sequence of intervals that are subsets of the sequence of sets that would arise from repeatedly applying $T_{D,\boldsymbol{\alpha}}$ to an initial open interval. Unless otherwise noted, assume that the function $m$ denotes the Lebesgue measure.

Definition 30. *Given an initial open interval $U \subset [0, 1)$ and $T_{D,\boldsymbol{\alpha}} \in \mathfrak{M}$, we* *define the infinite sequence of intervals* $\mathcal{U} = \{U_i\}_{i\in\mathbb{N}}$ *as*

$$U_1 = U \quad \textit{and}$$

$$U_{i+1} = \begin{cases} T_{D,\boldsymbol{\alpha}}(U_i) & : \quad U_i^\circ \subseteq [0,\frac{1}{2}) \qquad\qquad \textit{or } U_i^\circ \subseteq [\frac{1}{2},1), \\ T_{D,\boldsymbol{\alpha}}(U_i \cap [0,\frac{1}{2})) & : \quad U_i^\circ \not\subseteq [0,\frac{1}{2}) \textit{ and } U_i^\circ \not\subseteq [\frac{1}{2},1) \textit{ and} \\ & \qquad\qquad m(U_i \cap [0,\frac{1}{2})) \geq m(U_i \cap [\frac{1}{2},1)), \\ T_{D,\boldsymbol{\alpha}}(U_i \cap [\frac{1}{2},1)) & : \quad U_i^\circ \not\subseteq [0,\frac{1}{2}) \textit{ and } U_i^\circ \not\subseteq [\frac{1}{2},1) \textit{ and} \\ & \qquad\qquad m(U_i \cap [0,\frac{1}{2})) < m(U_i \cap [\frac{1}{2},1)). \end{cases}$$

DEFINITION 31 (critical points). *For a given* $T_{D,\boldsymbol{\alpha}}$ *where* $\boldsymbol{\alpha} \in \mathfrak{A}_n$*, define the set*

$$C = \{c_i \,:\, i \in \{1,\dots,m\}, c_i \in B \cup \{0,\tfrac{1}{2},1\}\}$$

*where* $B = \{b \,:\, \frac{1}{2} < b < 1$ *and* $b \in \{\alpha_1 D,\dots,\alpha_n D\} \cup \{\psi_1^x,\dots,\psi_{n-1}^x\}\}$ *and* $c_1 < c_2 < \dots < 1$. $C$ *is called the set of critical points for* $T_{D,\boldsymbol{\alpha}}$.

LEMMA 32 (doubling). *Given* $T_{D,\boldsymbol{\alpha}} \in \mathfrak{M}$*, let the sequence of intervals* $\mathcal{U}$ *be defined as in Definition* 30 *and let* $U_i$ *be some interval in the sequence. Furthermore, let* $C = \{c_1,\dots,c_m\}$ *be the set of critical points for* $T_{D,\boldsymbol{\alpha}}$. *If* $U_i \subseteq [c_j, c_{j+1}]$ *for some* $j \in \{1,\dots,m-1\}$*, then* $m(U_{i+1}) = 2m(U_i)$.

*Proof.* Since $U_i \subseteq [c_j, c_{j+1}]$ for some $j \in \{1,\dots,m-1\}$, because we are in the first case of the definition of $\mathcal{U}$, either $U_i^\circ \subseteq [0,\frac{1}{2})$ or $U_i^\circ \subseteq [\frac{1}{2},1)$. By simple inspection of the individual cases that define $T_{D,\boldsymbol{\alpha}}$, it is apparent that all of $U_i$, except possibly the points $c_j$ and $c_{j+1}$, fall within the same case of $T_{D,\boldsymbol{\alpha}}$. Therefore, the resulting interval $U_{i+1}$ will be double the length of $U_i$. □

DEFINITION 33 (active valleys). *Given* $T_{D,\boldsymbol{\alpha}} \in \mathfrak{M}_n$*, define*

$$V = \{\alpha_i D \,:\, i \in \{1,\dots,n\} \textit{ and } \tfrac{1}{2} < \alpha_i D < 1\}.$$

$V$ *is called the set of active valleys for* $T_{D,\boldsymbol{\alpha}}$.

DEFINITION 34 (active peaks). *Given* $T_{D,\boldsymbol{\alpha}} \in \mathfrak{M}_n$*, define*

$$P = \{\psi_i^x \,:\, i \in \{1,\dots,n-1\} \textit{ and } \tfrac{1}{2} < \psi_i^x < 1\}.$$

$P$ *is called the set of active peaks for* $T_{D,\boldsymbol{\alpha}}$.

LEMMA 35 (nonshrinking). *Given* $T_{D,\boldsymbol{\alpha}} \in \mathfrak{M}_n$ *with* $sep(\boldsymbol{\alpha}) \leq \frac{5}{3}$*, let the sequence of intervals* $\{U_i\}_{i\in\mathbb{N}}$ *be defined as above and let* $V$ *denote the set of active valleys for* $T_{D,\boldsymbol{\alpha}}$. *For any interval* $U_i \in \mathcal{U}$ *such that* $V \cap U_i = \varnothing$*, either* $m(U_{i+1}) \geq m(U_i)$ *or* $m(U_{i+2}) \geq m(U_i)$.

*Proof.* $sep(\boldsymbol{\alpha}) \leq \frac{5}{3}$ implies that $\alpha_{i+1} \leq \frac{5}{3}\alpha_i$. For a given separation, the value of $\psi_i^y$ is maximized when $\psi_i^x = 1$. This implies that $\alpha_i = \frac{3}{4D}$. We calculate the value of $\psi_i^y$ with the assumption that $\psi_i^x = 1$ to get a bound on $\psi_i^y$ for $D < 1$:

$$\psi_i^y \leq D(\tfrac{5}{3}\alpha_i - \alpha_i) = D(\tfrac{2}{3}\alpha_i) = D(\tfrac{2}{3}\tfrac{3}{4D}) = \tfrac{1}{2}.$$

*Case* 1. Consider when $U_i \subseteq [0,\frac{1}{2}]$. In this case, $m(U_{i+1}) = 2m(U_i)$.

*Case* 2. Consider when $U_i \subseteq [\frac{1}{2},1)$. The interval $U_i$ can span at most one peak. Therefore, $m(U_{i+1}) \geq m(U_i)$. A further observation is that since $U_{i+1} \subseteq [0,\frac{1}{2}]$, $m(U_{i+2}) = 2m(U_i)$.

*Case* 3. Consider when $U_i \not\subseteq [0, \frac{1}{2}]$ and $U_i \not\subseteq [\frac{1}{2}, 1)$. In this case, $U_i$ straddles $\frac{1}{2}$. From the definition of $\mathcal{U}$, we see that in the worst case we might throw away up to half of $U_i$. Call the part not thrown away $U_i{}'$ and observe that $m(U_i{}') \geq \frac{1}{2}m(U_i)$. Now, either $U_i{}' \subseteq [0, \frac{1}{2}]$ or $U_i{}' \subseteq [\frac{1}{2}, 1)$. If $U_i{}' \subseteq [0, \frac{1}{2}]$, then $m(U_{i+1}) = 2m(U_i{}') \geq m(U_i)$. If $U_i{}' \subseteq [\frac{1}{2}, 1)$, then $m(U_{i+2}) = 2m(U_i{}') \geq m(U_i)$. $\square$

LEMMA 36. *A multithreshold SRT division transformation $T_{D,\boldsymbol{\alpha}} \in \mathfrak{M}$ is expanding when sep$(\boldsymbol{\alpha}) \leq \frac{5}{3}$.*

*Proof.* Let $V$ be the set of active valleys (as defined in Definition 33) for a $T_{D,\boldsymbol{\alpha}}$. Let $P$ be the set of active peaks (as defined in Definition 34) for a $T_{D,\boldsymbol{\alpha}}$. Let $\mathcal{U} = \{U_i\}_{i \in \mathbb{N}}$ be the sequence of intervals associated with a $T_{D,\boldsymbol{\alpha}}$ and an initial interval $U$.

By way of contradiction, assume that a $T_{D,\boldsymbol{\alpha}}$ is not expanding. This means that for some $T_{D,\boldsymbol{\alpha}}$, there does not exist an interval $U_i$ where any of the points in $V$ are contained in $U_i$. This is true because if any of the valley points are in $U_i$, then $U_{i+1} = [0, \varepsilon)$ or $U_{i+1} = [0, \varepsilon]$, and after a finite number of steps, $U_i$ will have grown to include all of $[0, 1)$.

If there is a sequence $\mathcal{U}$ that avoids all points in $V$, then by Lemma 35 it must be true that the intervals in the sequence can double only a finite number of times. Let $i \in \mathbb{N}$ be the first index for which there is no $j > i$ such that $m(U_j) \geq 2m(U_i)$. It now follows that $U_i$ straddles $\frac{1}{2}$. The proof for Lemma 35 reveals that this is the only situation where it is not necessarily the case that either $m(U_{i+1}) = 2m(U_i)$ or $m(U_{i+2}) = 2m(U_i)$. In fact, $U_i$ must straddle both $\frac{1}{2}$ and $\min P$. If $\min P$ is not straddled and $m(U_i \cap [0, \frac{1}{2})) < m(U_i \cap [\frac{1}{2}, 1))$, then either $m(U_{i+2}) \geq 2m(U_i)$ or $m(U_{i+3}) \geq 2m(U_i)$. In the other possibility, where $\min P$ is not straddled and $m(U_i \cap [0, \frac{1}{2})) \geq m(U_i \cap [\frac{1}{2}, 1))$, we find that $m(U_{i+2}) \geq 2m(U_i)$.

Assuming that $U_i$ straddles both $\frac{1}{2}$ and $\min P$, we also observe that there can be no $j > i$ such that $m(U_j \cap [0, \frac{1}{2})) \geq m(U_j \cap [\frac{1}{2}, 1))$ because this quickly leads to doubling. In other words, the right side must be larger than the left side whenever we straddle $\frac{1}{2}$. Therefore, we must be in the situation where

$$\begin{aligned}
& U_i = (\tfrac{1}{2} - \varepsilon', \tfrac{1}{2} + \varepsilon),\ \varepsilon' < \varepsilon \\
\Rightarrow\quad & U_{i+1} = (\min\{2(\tfrac{1}{2} - \alpha_i D), 2(\alpha_{i+1} D - (\tfrac{1}{2} + \varepsilon))\}, \psi_i^y) \\
\Rightarrow\quad & U_{i+2} = (2\min\{2(\tfrac{1}{2} - \alpha_i D), 2(\alpha_{i+1} D - (\tfrac{1}{2} + \varepsilon))\}, 2\psi_i^y) \\
\Rightarrow\quad & U_{i+3} = (\min\{2(\tfrac{1}{2} - \alpha_i D), 2(\alpha_{i+1} D - 2\psi_i^y)\}, \psi_i^y) \\
\Rightarrow\quad & U_{i+4} = (2\min\{2(\tfrac{1}{2} - \alpha_i D), 2(\alpha_{i+1} D - 2\psi_i^y)\}, 2\psi_i^y) \\
\Rightarrow\quad & U_{i+5} = (\min\{2(\tfrac{1}{2} - \alpha_i D), 2(\alpha_{i+1} D - 2\psi_i^y)\}, \psi_i^y) = U_{i+3}\,.
\end{aligned}$$

It is apparent that the interval represented by $U_{i+4}$ will re-occur every other interval ad infinitum. We now use this interval to show that in fact such a sequence of nonexpanding intervals is not possible.

Since $U_{i+4}$ straddles $\frac{1}{2}$, we can compare the length of the left and right sides of $U_{i+4}$. Let $R = [\frac{1}{2}, 2\psi_i^y)$ denote the right side and let $L = (4(\frac{1}{2} - \alpha_i D), \frac{1}{2})$ and $L' = (4(\alpha_{i+1} D - 2\psi_i^y), \frac{1}{2})$ denote the two possibilities for the left side. The length of the right side is

$$m(R) = 2\psi_i^y - \tfrac{1}{2},$$

while the length of the left side is the larger of two possible lengths

$$m(L) = \tfrac{1}{2} - 4(\tfrac{1}{2} - \alpha_i D)$$

and

$$m(L') = \tfrac{1}{2} - 4(\alpha_{i+1}D - 2\psi_i^y) \ .$$

We then compare the differences between the right side and each of the two possible left sides. The first possibility is

$$\begin{aligned} m(R) - m(L) &= 2\psi_i^y - \tfrac{1}{2} - (\tfrac{1}{2} - 4(\tfrac{1}{2} - \alpha_i D)) \\ &= 2D(\alpha_{i+1} - \alpha_i) - 1 + 2 - 4\alpha_i D \\ &= 2\alpha_{i+1}D - 6\alpha_i D + 1 \ , \end{aligned}$$

while the second possibility is

$$\begin{aligned} m(R) - m(L') &= 2\psi_i^y - \tfrac{1}{2} - (\tfrac{1}{2} - 4(\alpha_{i+1} - 2\psi_i^y)) \\ &= 2D(\alpha_{i+1} - \alpha_i) - 1 + 4(\alpha_{i+1}D - 2D(\alpha_{i+1} - \alpha_i)) \\ &= -2\alpha_{i+1}D + 6\alpha_i D - 1 \, . \end{aligned}$$

It is now clear that

$$m(R) - m(L) = -\left(m(R) - m(L')\right) \ .$$

But this means that the length of the left side is always greater than or equal to the length of the right side, which contradicts our assumption that the right side must be bigger than the left side whenever the interval straddles $\frac{1}{2}$. □

THEOREM 37. *$T_{D,\boldsymbol{\alpha}} \in \mathfrak{M}$ is Bernoulli when sep$(\boldsymbol{\alpha}) \leq \frac{5}{3}$.*

*Proof.* Let $T = T_{D,\boldsymbol{\alpha}}$. From the definition of $T$, we see that $T_{D,\boldsymbol{\alpha}}$ is $C^2$ and that $\inf_{0 \leq x \leq 1} |T'(x)| = 2 > 1$ since $|T'(x)| = 2$ for all $x$ for which the derivative is defined. Since $\inf_{0 \leq x \leq 1} |T'(x)| > 1$, by Theorem 11, there exists at least one $\mu$ such that $\mu$ is a smooth $T$-invariant probability measure. By Lemma 36 we see that Theorem 10 holds when *sep*$(\boldsymbol{\alpha}) \leq \frac{5}{3}$. Hence, $(T, \mu)$ is weak-mixing and, by Theorem 9, $(T, \mu)$ is Bernoulli when *sep*$(\boldsymbol{\alpha}) \leq \frac{5}{3}$. □

The calculation for entropy in multithreshold SRT division follows the same method used for single divisor SRT division. We begin by showing that $T_{D,\boldsymbol{\alpha}}$ is a PC-map.

LEMMA 38. *$T_{D,\boldsymbol{\alpha}} \in \mathfrak{M}$ is a PC-map (as defined in Definition* 20*).*

*Proof.* By inspection, each $T_{D,\boldsymbol{\alpha}}$ is a finite collection of line segments, each with slope 2. Each of these line segments is a C-map by the same argument used in the proof for Theorem 22. Therefore, by definition, each $T_{D,\boldsymbol{\alpha}}$ is a PC-map. □

THEOREM 39. *The entropy of any $T_{D,\boldsymbol{\alpha}} \in \mathfrak{M}$ with sep$(\boldsymbol{\alpha}) \leq \frac{5}{3}$ is* $\log 2$.

*Proof.* By Lemma 38, all $T_{D,\boldsymbol{\alpha}} \in \mathfrak{M}$ are PC-maps. By Theorem 37, $T_{D,\boldsymbol{\alpha}}$ is Bernoulli when *sep*$(\boldsymbol{\alpha}) \leq \frac{5}{3}$, and hence there exists a unique a.c.i.m. $\mu$. Theorem 21 says that Rohlin's formula for the entropy is true, and therefore

$$h(T_{D,\boldsymbol{\alpha}}) = \int \log \left|T_{D,\boldsymbol{\alpha}}{}'\right| \mathrm{d}\mu = \log 2 \int \mathrm{d}\mu = \log 2. \qquad \square$$

**5. Some restrictions on $\boldsymbol{\alpha}$ for multithreshold division.** In section 4, we showed that for all $T_{D,\boldsymbol{\alpha}} \in \mathfrak{M}$, if *sep*$(\alpha) \leq \frac{5}{3}$, then $T_{D,\boldsymbol{\alpha}}$ is Bernoulli. In this section, we construct examples of $T \in \mathfrak{M}_n$, for every $n$, that fail to be Bernoulli when the restriction that *sep*$(\boldsymbol{\alpha}) \leq \frac{5}{3}$ is relaxed.

THEOREM 40. *For $T_{D,\boldsymbol{\alpha}} \in \mathfrak{M}_{n \geq 4}$, if sep$(\boldsymbol{\alpha}) > \frac{5}{3}$, then for each $D \in [\frac{1}{2}, 1)$, there exist uncountably many $\boldsymbol{\alpha}$ for which $T_{D,\boldsymbol{\alpha}}$ is not ergodic.*

*Proof.* We begin this proof by considering $T \in \mathfrak{M}_{n=4}$.

Assume that we relax the restrictions on $\boldsymbol{\alpha}$ by $\varepsilon > 0$. This means that $sep(\boldsymbol{\alpha}) \leq \frac{5}{3} + \varepsilon$ and that no peak can be above the line $f(x) = \frac{4+6\varepsilon x}{8+3\varepsilon}$. With this relaxation, we can define $\boldsymbol{\alpha} = (\alpha_1, \alpha_2, \alpha_3, \alpha_4)$ with respect to a given $D$ so that a subset of $[0, 1)$ is nonexpanding. We let $\alpha_1 = \frac{30+27\varepsilon}{80D+48D\varepsilon}$, $\alpha_2 = \frac{50+57\varepsilon}{80D+48D\varepsilon}$, $\alpha_3 = \frac{30-9\varepsilon}{40D+24D\varepsilon}$, and $\alpha_4 = \frac{50+21\varepsilon}{40D+24D\varepsilon}$. For our constructed $\boldsymbol{\alpha}$ to be valid, we must be careful that conditions (a) and (b) of Definition 23 hold. Condition (a) requires that the components of $\boldsymbol{\alpha}$ remain in ascending order. This is satisfied when $\varepsilon \in (0, \frac{2}{15}]$. Since ordering is maintained, $sep(\boldsymbol{\alpha}) < 3$, and $\min_{D\in[1/2,1),\, \varepsilon\in(0,2/15]} \alpha_4 = 1.\overline{2} \geq 1$, to verify that condition (b) of Definition 23 holds, it is sufficient to show (by Lemma 28) that for all values of $D$ and $\varepsilon$, either $\alpha_1$, $\alpha_2$, or $\alpha_3 \in [\frac{1}{2}, 1]$. By maximizing and minimizing over $\varepsilon$ and $D$, we find that $\alpha_1 \in [0.375, 0.\overline{7}]$ and $\alpha_2 \in [0.625, 1.\overline{3}]$. Figure 6 provides a visual proof that as $\varepsilon$ is varied over $[0, \frac{2}{15}]$ and $D$ is varied over $[\frac{1}{2}, 1]$, it is never the case that both $\alpha_1 \leq \frac{1}{2}$ and $\alpha_2 \geq 1$. Therefore, it is always the case that either $\alpha_1$ or $\alpha_2 \in [\frac{1}{2}, 1]$.

Having verified that our defined $\boldsymbol{\alpha}$ satisfies Definition 23, we calculate that peak $\boldsymbol{\psi}_1 = (\frac{20+21\varepsilon}{40+24\varepsilon}, \frac{10+15\varepsilon}{40+24\varepsilon})$ and peak $\boldsymbol{\psi}_3 = (\frac{20+3\varepsilon}{20+12\varepsilon}, \frac{10+15\varepsilon}{20+12\varepsilon})$. With this definition for $\boldsymbol{\alpha}$, and our assumption that $\varepsilon \in [0, \frac{2}{15})$, the point $\boldsymbol{\psi}_3$ will always touch the line $f(x)$ while remaining above the line $g(x) = \frac{1}{2}$, and the point $\boldsymbol{\psi}_1$ will always be slightly below $f(x)$ while remaining above the line $g(x) = \frac{1}{4}$. All of the definitions have been chosen so that $1 - \psi_3^x = \psi_3^y - \frac{1}{2} = 2(\psi_1^x - \frac{1}{2}) = 2(\psi_1^y - \frac{1}{4})$. Another important feature in this construction is the interval between $\alpha_2 D$ and $\alpha_3 D$. Since $\boldsymbol{\psi}_2$ is not used in our construction, it is possible to insert an arbitrary number of divisor multiples between $\alpha_2 D$ and $\alpha_3 D$. Thus, the results in this proof apply to $T \in \mathfrak{M}_n$ for arbitrarily large $n$.

We are now in a position to show that there exists a set of points $A$ with $0 < m(A) < 1$, for which $T_{D,\boldsymbol{\alpha}}(A) = A$. This is the definition of a transformation being nonergodic [8, p. 59]. Define $A = A_1 \cup A_2 \cup A_3$, where $A_1 = [\frac{1}{4} - (\psi_1^x - \frac{1}{2}), \frac{1}{4} + (\psi_1^x - \frac{1}{2})]$, $A_2 = [\frac{1}{2} - 2(\psi_1^x - \frac{1}{2}), \frac{1}{2} + 2(\psi_1^x - \frac{1}{2})]$, and $A_3 = [1 - 2(1 - \psi_3^x), 1]$. It can be shown by calculation that $T_{D,\boldsymbol{\alpha}}(A_1) = A_2$, $T_{D,\boldsymbol{\alpha}}(A_2) = A_1 \cup A_3$, and $T_{D,\boldsymbol{\alpha}}(A_3) = A_2$. Therefore, $T_{D,\boldsymbol{\alpha}}(A) = A$, and by definition, $T_{D,\boldsymbol{\alpha}}$ is nonergodic or nonexpanding. ∎

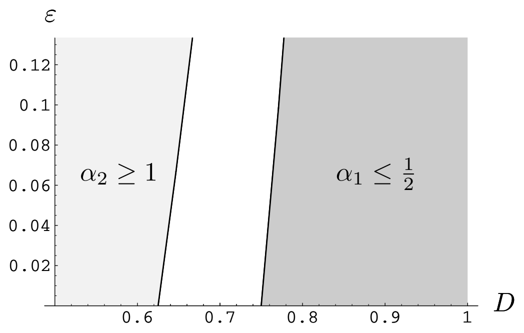


FIG. 6. *Combined plot of the regions where* $\alpha_1 \leq \frac{1}{2}$ *and* $\alpha_2 \geq 1$.

Figure 7 illustrates the type of transformation that we have constructed in the proof of Theorem 40. In this figure, $n = 4$, $D = \frac{11}{16}$, $\boldsymbol{\alpha} = (\frac{37}{66}, \frac{21}{22}, 1, \frac{59}{33})$, and $sep(\boldsymbol{\alpha}) = \frac{5}{3} + \frac{5}{51}$. The thick lines represent $T_{D,\boldsymbol{\alpha}}$. The coarse dashed line represents the necessary separation restriction on $\boldsymbol{\alpha}$ to guarantee that $T_{D,\boldsymbol{\alpha}}$ is ergodic. In this case, partial remainders in the set $A = [\frac{11}{48}, \frac{13}{48}] \cup [\frac{22}{48}, \frac{26}{48}] \cup [\frac{44}{48}, 1)$ are mapped back to $A$ by $T_{D,\boldsymbol{\alpha}}$.

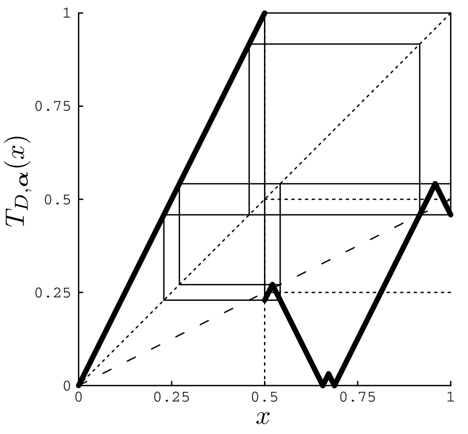


FIG. 7. *An example of a nonergodic system for $T_{D,\boldsymbol{\alpha}} \in \mathfrak{M}_{n\geq 4}$.*

THEOREM 41. *For $T_{D,\boldsymbol{\alpha}} \in \mathfrak{M}_3$, if $sep(\boldsymbol{\alpha}) \geq \frac{9}{5}$, then for each $D \in [\frac{1}{2}, 1)$, there exists an $\boldsymbol{\alpha}$ for which $T_{D,\boldsymbol{\alpha}}$ is not ergodic.*

*Proof.* The proof for this theorem comes as a special case from the proof for Theorem 40. Consider $\boldsymbol{\alpha} = (\alpha_1, \alpha_2, \alpha_3, \alpha_4)$ as defined in the proof for Theorem 40. When $sep(\boldsymbol{\alpha}) = \frac{9}{5} = \frac{5}{3} + \frac{2}{15}$, we are in the special situation where $\alpha_2 = \alpha_3$. Since all of the results for the proof of Theorem 40 still hold, we now have an example transformation $T$ with only three unique multiples of $D$, and this $T$ has been proven to be nonergodic. $\square$

Figure 8 gives an example of a nonergodic transformation for $D = \frac{7}{12}$ and $\boldsymbol{\alpha} = (\frac{2}{3}, \frac{8}{7}, \frac{44}{21})$. In this case, partial remainders in the set $A = [\frac{4}{18}, \frac{5}{18}] \cup [\frac{8}{18}, \frac{10}{18}] \cup [\frac{16}{18}, 1)$ are mapped back to $A$ by $T_{D,\boldsymbol{\alpha}}$.

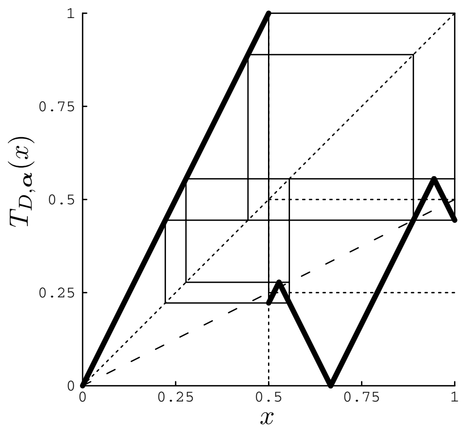


FIG. 8. *An example of a nonergodic system for $T_{D,\boldsymbol{\alpha}} \in \mathfrak{M}_3$.*

THEOREM 42. *For $T_{D,\boldsymbol{\alpha}} \in \mathfrak{M}_2$, if $sep(\boldsymbol{\alpha}) > 3$, then for some $D \in (\frac{1}{2}, 1)$, there exist uncountably many $\boldsymbol{\alpha}$ for which $T_{D,\boldsymbol{\alpha}}$ is not ergodic.*

*Proof.* Assume that $sep(\boldsymbol{\alpha}) \leq 3 + \varepsilon$ and $D \in (\frac{1}{2}, \frac{2+\varepsilon}{4})$. First, we choose $\alpha_1 = \frac{1}{4D}$ so that $\alpha_1 D = \frac{1}{4}$ and $\alpha_2 = 1 + \alpha_1$. Our restriction on $D$ in terms of $\varepsilon$ has been chosen so that $\alpha_2/\alpha_1 < 3 + \varepsilon$ when $\alpha_2 = 1 + \alpha_1$. Since $\alpha_2 > \alpha_1$, condition (a) of Definition

23 is satisfied. Since $\alpha_1 \in (\frac{1}{4}, \frac{1}{2})$ and $\alpha_2 \in (1, 1 + \alpha_1]$ by Lemma 28, condition (b) of Definition 23 is satisfied. Thus, our defined $\boldsymbol{\alpha}$ is always valid. Define $A = [\frac{1}{2}, D]$. We now apply $T = T_{D,\boldsymbol{\alpha}}$ to $A$:

$$\begin{aligned} T[\tfrac{1}{2}, D] &= [\min\{2(\tfrac{1}{2} - \alpha_1 D), 2(\alpha_2 D - D)\}, \psi_1^y] \\ &= [\min\{2(\tfrac{1}{2} - \tfrac{D}{4D}), 2(D + \tfrac{1}{4} - D)\}, D(\alpha_2 - \alpha_1)] \\ &= [\min\{\tfrac{1}{2}, \tfrac{1}{2}\}, D(1 + \tfrac{1}{4D} - \tfrac{1}{4D})] \\ &= [\tfrac{1}{2}, D]\,. \end{aligned}$$

Now, since $\frac{1}{2} < D < 1$, $0 < m(A) < 1$, and $T_{D,\boldsymbol{\alpha}}A = A$, by definition $T_{D,\boldsymbol{\alpha}}$ is not ergodic. ◻

Figure 9 shows an example of a nonergodic system for $T_{D,\boldsymbol{\alpha}} \in \mathfrak{M}_2$. In this example, $D = \frac{3}{5}$ and $\boldsymbol{\alpha} = (\frac{5}{12}, \frac{17}{12})$. Partial remainders within the interval $[\frac{1}{2}, \frac{3}{5}]$ map back to $[\frac{1}{2}, \frac{3}{5}]$.

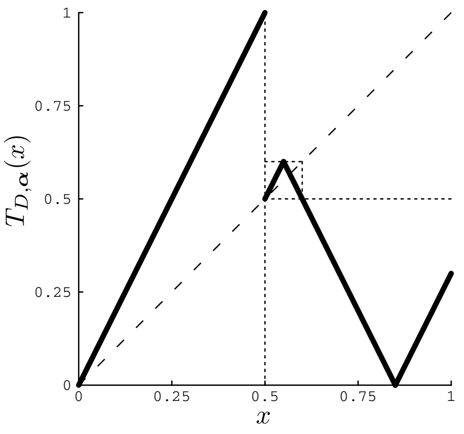


FIG. 9. *An example of a nonergodic system for* $T_{D,\boldsymbol{\alpha}} \in \mathfrak{M}_2$.

**6. Conclusions.** The original question that inspired this paper was, "Is simple SRT division ergodic for all real divisors?" In pursuing the solution to this problem, we discovered that not only is simple SRT division ergodic for all divisors, but it is also Bernoulli. Having established a Bernoulli property, and having calculated the entropy for our transformations, we were able to use the Kolmogorov–Ornstein theorem to conclude that our transformations are equivalent to each other in the sense that their natural extensions are isomorphic. In proving these important properties for simple SRT division, we made extensive use of more general results from dynamical systems theory. Consequently, our results were shown to be easily extensible to more general division systems. In general, it is difficult to prove that a particular class of transformations is ergodic or Bernoulli. Our results have provided an effective means of proving both of these properties for a large class of SRT-like division algorithms.

From the standpoint of understanding an algorithm's expected performance, it is necessary to know that when a stationary distribution is found, it is unique. Having established the uniqueness of stationary distributions, the next step is to find the actual stationary distribution for as wide a class of transformations as possible. In section 2, we gave the stationary distribution function for $T_D$, where $D \in [\frac{3}{4}, 1)$. Many of the stationary distribution functions have been classified by Shively and Freiman for $D \in [\frac{3}{5}, \frac{3}{4}]$, although the derivations are not nearly as simple as for $D \in [\frac{3}{4}, 1)$. It turns out that things become very complicated when $D \in [\frac{1}{2}, \frac{3}{5}]$. In his thesis [20], Shively shows many interesting properties for the stationary distribution functions in this region. For example, he shows that there are many different intervals of $D$, where there are an infinite number of different stationary distribution equations. As such, the graph of the shift average for $D \in [\frac{1}{2}, \frac{3}{5}]$ is far from complete and appears to have a complex pattern (from the few points that have been plotted in this region). This is surprising, considering the simplicity of the underlying transformation. A better understanding of this final region of simple SRT division would be an interesting goal to pursue.

In the work of Freiman [5], it was first shown that the shift average for $D \in [\frac{3}{5}, \frac{3}{4}]$ is constantly 3, which can be easily shown to be the maximum possible shift average. This property was then used by Metze [12] to obtain a version of SRT division that has an expected shift average of 3 for all divisors. Another area to pursue would be to explore shift averages for multithreshold SRT division and, if other plateaus are found, they could possibly be used to obtain higher expected shift averages for all possible divisors. Undoubtedly, obtaining a complete understanding of the stationary distribution functions for multithreshold division would be even more difficult than it is for simple SRT division. It is possible that such results in this area could lead to improvements in modern SRT division. Related to this, it would be interesting to attempt to extend the results of this paper to modern SRT division.

**Appendix. SRT division is chaotic.** In section 2 we mentioned that SRT division is chaotic, and we prove this fact here for SRT division with the expansion property (see Definition 6). Simple SRT division was shown to be expanding for all divisors (see Theorem 8). Multithreshold SRT division, of which simple SRT division is a special case, was shown to be expanding when there is a restriction on the separation of the divisor multiples (see Lemma 36).

Although there are several definitions for what it means to be "chaotic," the one given by Devaney [4] is commonly used. The following definitions are taken from [4, pp. 49, 50].

DEFINITION 43 (sensitivity). *A transformation $f : X \to X$ has sensitive dependence on initial conditions if there exists $\delta > 0$ such that, for any $x \in X$ and any neighborhood $N$ about $x$, there exists $y \in X$ and $n \geq 0$ such that $|f^n(x) - f^n(y) > \delta|$.*

DEFINITION 44 (topological transitivity). *A transformation $f : X \to X$ is topologically transitive on $X$ if, for any pair of open sets $U$, $V \subset X$, there exists $n$ such that $f^n(U) \cap V \neq \emptyset$.*

DEFINITION 45 (chaotic). *Let $f : X \to X$ be a transformation of the set $X$. $f$ is said to be chaotic on $X$ if*

(a) *$f$ has sensitive dependence on initial conditions;*

(b) *$f$ is topologically transitive;*

(c) *periodic points are dense in $X$.*

THEOREM 46. *Every multithreshold SRT division transformation with the expansion property is chaotic.*

*Proof.* Let $T$ be a multithreshold SRT division transformation on $[0, 1)$ having the expansion property.

To prove sensitivity to initial conditions, we notice, by the expansion property, that for any open neighborhood $N$ about a point $x$, the successive images of $N$ under $T$ expand to be arbitrarily close to filling the entire interval densely after a finite number $n$ of steps. In particular, since $f^n(x)$ must be at least distance $\frac{1}{2}$ away from either 0 or 1, for any $\frac{1}{2} > \delta > 0$, there is $y \in N$ such $|f^n(x) - f^n(y)| > \delta$. Therefore, $T$ is sensitive to initial conditions on $[0, 1)$.

Again, by the expansion property of $T$, the successive images of any two open intervals will intersect within a finite number of steps. Therefore, $T$ is topologically transitive on $[0, 1)$.

The expansion property alone is not sufficient to ensure that periodic points are dense. However, the existence of a particular sequence $\mathcal{U}$ of nonempty subintervals as given in Definition 30 is sufficient, in general, for dense periodic points.

Let $V_1$ be an open interval on part of the domain where $T$ is continuous. In the proof of Lemma 36, we showed that sequence $\{T^i(V_1)\}_{i=1}^{\infty}$ necessarily expands after a finite number of steps to fill the unit interval by proving that another sequence $\mathcal{U} = \{U_i\}$ of intervals expands to fill the unit interval where $U_i \subseteq V_i$ for $i \geq 0$. For a given $U_i$, $U_{i+1}$ is chosen to be the largest subinterval of $T(U_i)$ on a continuous part of $T$'s domain. In the event of equal-length subintervals, we choose the left half to be $U_i$'s successor. $\mathcal{U}$ induces another (possibly not unique) sequence of intervals $\{I_i\}$, where $I_i$ is any subinterval of $U_i$, where $T(I_i) = U_{i+1}$. In any situation where part of $T(U_i)$ is discarded, there exists at least one interval $I_i \subset U_i$, where $T(I_i) = U_{i+1}$ because $U_i$ is chosen such that $T$ is continuous on $U_i$. Note that $T|_{I_i} : I_i \to U_{i+1}$ is a continuous onto map. Eventually, by the expansion property, it must be true that $U_n \supseteq U_1 = V_1$ for some $n$. We now have a sequence of continuous onto maps $T|_{U_1} : I_1 \to U_2$, $T|_{U_2} : I_2 \to U_3$, $\ldots$, $T|_{U_{n-1}} : I_{n-1} \to U_n$ and it follows that there is a nonempty interval $U' \subseteq U_1$ such that $T^{n-1}(U') = T|_{U_{n-1}} \circ T|_{U_{n-2}} \circ \cdots \circ T|_{U_1} (U') = U_n$. Since $T^{n-1}|_{U'}$ is continuous, there exists $x \in V_1$ such that $T^{n-1}(x) = x$. $\square$

## REFERENCES

[1] D. E. Atkins, *Higher-radix division using estimates of the divisor and partial remainders*, IEEE Trans. Comput., C-17 (1968), pp. 925–934.

[2] R. Bowen, *Bernoulli maps of the interval*, Israel J. Math., 28 (1977), pp. 161–168.

[3] J. Cocke and D. W. Sweeney, *High Speed Arithmetic in a Parallel Device*, Tech. report, IBM, February 1957.

[4] R. L. Devaney, *An Introduction to Chaotic Dynamical Systems*, 2nd ed., Westview Press, Boulder, CO, 2003.

[5] C. V. Freiman, *Statistical analysis of certain binary division algorithms*, Proc. IRE, 49 (1961), pp. 91–103.

[6] A. N. Kolmogorov, *A new metric invariant for transient dynamical systems and automorphisms in Lebesgue spaces*, Dokl. Akad. Nauk SSSR (N.S.), 119 (1958), pp. 861–864 (in Russian).

[7] A. N. Kolmogorov, *Entropy per unit time as a metric invariant of automorphisms*, Dokl. Akad. Nauk SSSR, 124 (1959), pp. 754–755 (in Russian).

[8] A. Lasota and M. C. Mackey, *Chaos, Fractals, and Noise: Stochastic Aspects of Dynamics*, 2nd ed., Springer-Verlag, New York, 1994.

[9] A. Lasota and J. A. Yorke, *On the existence of invariant measures for piecewise monotonic transformations*, Trans. Amer. Math. Soc., 186 (1973/1974), pp. 481–488.

[10] F. Ledrappier, *Some properties of absolutely continuous invariant measures on an interval*, Ergodic Theory Dynamical Systems, 1 (1981), pp. 77–93.

[11] M. McCann and N. Pippenger, *SRT division algorithms as dynamical systems*, in Proceedings of the 16th IEEE Symposium on Computer Arithmetic, 2003, pp. 46–53.

[12] G. Metze, *A class of binary divisions yielding minimally represented quotients*, IRE Trans. Electron. Comput., EC-11 (1961), pp. 761–764.

[13] M. Nadler, *A high-speed electronic arithmetic unit for automatic computing machines*, Acta Tech. Prague, 1 (1956), pp. 464–478.

[14] D. S. Ornstein, *Bernoulli shifts with the same entropy are isomorphic*, Advances in Math., 4 (1970), pp. 337–352.

[15] D. S. Ornstein, *Ergodic Theory, Randomness, and Dynamical Systems*, James K. Whittemore Lectures in Mathematics given at Yale University, Yale Math. Monographs 5, Yale University Press, New Haven, CT, 1974.

[16] K. Petersen, *Ergodic Theory*, Cambridge University Press, Cambridge, UK, 1983.

[17] J. E. Robertson, *A new class of digital division methods*, IRE Trans. Electron. Comput., EC-7 (1958), pp. 218–222.

[18] V. A. Rohlin, *Exact endomorphisms of a Lesbesgue space*, Amer. Math. Soc. Transl., 39 (1964), pp. 1–36.

[19] P. Shields, *The Theory of Bernoulli Shifts*, Chicago Lectures in Math., The University of Chicago Press, Chicago, IL, London, 1973.

[20] R. Shively, *Stationary Distribution of Partial Remainders in S-R-T Digital Division*, Ph.D. thesis, University of Illinois, Chicago, 1963.

[21] K. D. Tocher, *Techniques of multiplication and division for automatic binary computers*, Quart. J. Mech. Appl. Math., 11 (1958), pp. 364–384.

[22] P. Walters, *An Introduction to Ergodic Theory*, Springer-Verlag, New York, 1982.

SIAM J. Comput. © 2005 Society for Industrial and Applied Mathematics
Vol. 34, No. 6, pp. 1302–1323

# POLYNOMIAL-TIME APPROXIMATION SCHEMES FOR GEOMETRIC INTERSECTION GRAPHS[*]

THOMAS ERLEBACH[†], KLAUS JANSEN[‡], AND EIKE SEIDEL[‡]

**Abstract.** A disk graph is the intersection graph of a set of disks with arbitrary diameters in the plane. For the case that the disk representation is given, we present polynomial-time approximation schemes (PTASs) for the maximum weight independent set problem (selecting disjoint disks of maximum total weight) and for the minimum weight vertex cover problem in disk graphs. These are the first known PTASs for $\mathcal{NP}$-hard optimization problems on disk graphs. They are based on a novel recursive subdivision of the plane that allows applying a shifting strategy on different levels simultaneously, so that a dynamic programming approach becomes feasible. The PTASs for disk graphs represent a common generalization of previous results for planar graphs and unit disk graphs. They can be extended to intersection graphs of other "disk-like" geometric objects (such as squares or regular polygons), also in higher dimensions.

**Key words.** independent set, vertex cover, shifting strategy, disk graph

**AMS subject classifications.** 68Q25, 68R10

**DOI.** 10.1137/S0097539702402676

## 1. Introduction.

Intersection graphs are graphs whose vertices are represented by sets such that two vertices are adjacent if and only if the corresponding sets have a nonempty intersection. They have been studied by many authors [12, 6, 24]. We are interested in approximation algorithms for $\mathcal{NP}$-hard optimization problems on intersection graphs of geometric objects, in particular approximation algorithms for independent set and vertex cover. Two prominent applications of geometric intersection graphs are frequency assignment in cellular networks [13, 22] and map labeling [1].

The goal of the *maximum weight independent set* problem (MWIS) is to compute, for a given set of geometric objects with certain weights, a subset of disjoint (nonoverlapping) objects with maximum total weight. The goal of the *minimum weight vertex cover* problem (MWVC) is to compute a subset of the given objects with minimum total weight such that, for any two intersecting objects, at least one of the objects is contained in the subset. MIS and MVC refer to the unweighted versions of these problems. We obtain polynomial-time approximation schemes for MWIS and MWVC in the intersection graphs of disks, squares, or other "disk-like" objects, also in higher dimensions.

### 1.1. Preliminaries.

For a set $V$ of geometric objects, the corresponding *intersection graph* is the undirected graph with vertex set $V$ and an edge between two

[*]Received by the editors February 15, 2002; accepted for publication (in revised form) August 2, 2004; published electronically June 27, 2005. A preliminary version of this paper was presented at the 12th Annual ACM–SIAM Symposium on Discrete Algorithms (SODA 2001). This research was supported in part by DFG Graduiertenkolleg 357, Effiziente Algorithmen und Mehrskalenmethoden, by EU Research Training Network ARACNE, Approximation and Randomized Algorithms in Communication Networks, HPRN-CT-2000-00112, and by EU Thematic Network APPOL I + II, Approximation and Online Algorithms, IST-1999-14084 and IST-2001-32007.

http://www.siam.org/journals/sicomp/34-6/40267.html

[†]Department of Computer Science, University of Leicester, University Road, Leicester LE1 7RH, UK (t.erlebach@mcs.le.ac.uk).

[‡]Institute for Computer Science and Applied Mathematics, Christian-Albrechts-University of Kiel, Olshausenstr. 40, D-24098 Kiel, Germany (kj@informatik.uni-kiel.de, ese@informatik.uni-kiel.de).

vertices if the corresponding objects intersect.

Assume that we are given a set $\mathcal{D} = \{D_1, \ldots, D_n\}$ of $n$ (topologically closed) disks in the plane, where $D_i$ has diameter $d_i$, center $c_i = (x_i, y_i)$, and weight $w_i$. For a subset $U \subseteq \mathcal{D}$, $w(U)$ denotes the sum of the weights of the disks in $U$. Disks $D_i$ and $D_j$ *intersect* if $dist(c_i, c_j) \leq (d_i + d_j)/2$, where $dist(p_1, p_2)$ denotes the Euclidean distance between two points $p_1$ and $p_2$ in the plane. A *disk graph* is the intersection graph of a set of disks. We assume that the input to our algorithms is the set $\mathcal{D}$ of disks, not only the corresponding intersection graph. This is an important distinction, because determining for a given graph whether it is a disk graph is known to be $\mathcal{NP}$-hard [16], and hence no efficient method is known for computing a disk representation if only the intersection graph is given.

Interestingly, every planar graph is a *coin graph*, i.e., the intersection graph of a set of interior-disjoint disks [21]. Therefore, the class of disk graphs properly contains the class of planar graphs.

For a given set $\mathcal{D}$ of disks in the plane, we let $OPT_{IS}(\mathcal{D})$ and $OPT_{VC}(\mathcal{D})$ denote the total weight of an optimal solution for MWIS and MWVC, respectively. An algorithm is a $\rho$-approximation algorithm for MWIS if it runs in polynomial time and always computes an independent set of total weight at least $\frac{1}{\rho} OPT_{IS}(\mathcal{D})$. An algorithm is a $\rho$-approximation algorithm for MWVC if it runs in polynomial time and always computes a vertex cover of total weight at most $\rho OPT_{VC}(\mathcal{D})$. An algorithm is a *polynomial-time approximation scheme* (PTAS) for MWIS if it takes an additional parameter $\varepsilon > 0$ and always computes an independent set of total weight at least $\frac{1}{1+\varepsilon} OPT_{IS}(\mathcal{D})$, where the running-time is polynomial in the size of the representation of $\mathcal{D}$ for fixed $\varepsilon > 0$. A PTAS for MWVC is defined analogously. If an algorithm is a $\rho$-approximation algorithm, the algorithm is also said to have approximation ratio $\rho$.

Note that the complement of an independent set is a vertex cover, and vice versa. Therefore, we have $OPT_{IS}(\mathcal{D}) = w(\mathcal{D}) - OPT_{VC}(\mathcal{D})$. Nevertheless, taking the complement of the solution output by a $\rho$-approximation algorithm for MWIS does in general not provide a $\rho$-approximation for MWVC, and vice versa.

In general graphs with $n$ vertices, there cannot be a polynomial-time approximation algorithm for MWIS with approximation ratio $n^{1-\varepsilon}$ for any $\varepsilon > 0$ unless $\mathcal{NP} = \text{co-}RP$ [14]. MWVC is MAX SNP-hard in general graphs and cannot be approximated within a constant smaller than 7/6 unless $\mathcal{P} = \mathcal{NP}$ [15]. For MWVC in general graphs, a 2-approximation algorithm is known [3]. For intersection graphs of geometric objects, better approximation ratios are often possible.

**1.2. Related work on disk graphs.** For unit disk graphs (intersection graphs of disks with equal diameter), MWIS and MWVC remain $\mathcal{NP}$-hard [9], but PTASs exist for MWIS, MWVC, and the minimum dominating set problem if the disk representation is given as part of the input [18]. For intersection graphs of disks with arbitrary diameters, the best previously known approximation algorithms achieve approximation ratio 5 for MIS [23] and $\frac{3}{2}$ for MWVC [22]. In [18] and [22], the question was raised whether a PTAS exists for disk graphs. As the class of disk graphs contains the class of unit disk graphs and the class of planar graphs, a PTAS for disk graphs would generalize the results for unit disk graphs due to Hunt et al. [18] and the results for planar graphs due to Baker [2]. This paper resolves this question by presenting PTASs for MWIS and MWVC in disk graphs (with given representation).

A PTAS for the fractional chromatic number problem on disk graphs, using our PTAS for MWIS as a subroutine, was obtained in [20, 19].

**1.3. Related work on map labeling.** Map labeling refers to a family of tasks concerning the placement of labels on a map. Often it is assumed that the features to be labeled are points and that the labels can be modeled as rectangles (just take the bounding box of the label text). For each feature, there are certain admissible positions of the labeling rectangle: for example, the rectangle must be placed so that the feature coincides with a corner of the rectangle. Rectangles can be assigned weights that represent the importance of including that label on the map. Then it is meaningful to study the problem of maximizing the total weight of labeled features subject to the constraint that different labels must not overlap. This is just MWIS in the intersection graph of a set of (topologically closed) axis-aligned rectangles.

Agarwal, van Kreveld, and Suri [1] give an $O(\log n)$-approximation algorithm for MIS in an intersection graph of $n$ axis-aligned rectangles. Their algorithm can be adapted to the weighted case in a straightforward way, thus yielding an approximation ratio of $O(\log n)$ also for MWIS. For the special case that all rectangles have the same height (which is meaningful if the labels are text labels of a certain font size), they obtain a PTAS.

Doddi et al. [10] consider *sliding labels* (a point can lie anywhere on the boundary of its label) and assume that labels may be placed in any orientation. They provide constant-factor approximation algorithms for maximizing the size of the labels (assuming that all features must be labeled and that all labels are circles or squares with identical size). These were recently improved in [11]. In [10], bicriteria approximation algorithms that label a $(1-\varepsilon)$-fraction of all features with labels whose size is at least a $\frac{1}{1+\varepsilon}$-fraction of the optimal size are also discussed. These algorithms use a PTAS for MWIS as a subroutine. In [10], it is mentioned that their algorithms can be extended to the case of nonuniform squares if the ratio between the size of the largest square and the smallest square is bounded by a constant. Using our new PTAS for MWIS in the intersection graphs of squares, their algorithms can now be extended to labeling with nonuniform squares where this ratio is arbitrary.

Van Kreveld, Strijk, and Wolff [27] investigate the question of how many features in a map can be labeled with rectangular labels if different restrictions on the labeling model are enforced (feature must be at a corner of the rectangle versus sliding rectangles). They present a practical 2-approximation algorithm and a PTAS for maximizing the number of labeled features in the case of sliding rectangular labels of equal height.

An up-to-date bibliography of publications on map labeling can be found on the Web [28].

**1.4. Our results.** In this paper we present PTASs for MWIS and MWVC in the intersection graphs of "disk-like" objects. In sections 2 and 3, we present the details of the PTASs for MWIS and MWVC in disk graphs. In section 4, we discuss how our approach can be extended to other geometric objects (such as squares or regular polygons) and to higher dimensions. We give our conclusions and mention some open problems in section 5.

Our PTASs are based on a sophisticated use of the *shifting strategy* [2, 17] that was previously employed, among other results, for obtaining PTASs for various optimization problems in planar graphs [2] and unit disk graphs [18]. We partition the given disks into levels according to their diameters and use a novel recursive subdivision of the plane that allows us to apply the shifting strategy on all levels simultaneously.

We outline the basic idea of the PTAS for MWIS. The plane is partitioned into squares on each level, and some of the disks are removed from the input so that different squares on the same level yield independent subproblems with respect to all disks that are on this level or on a level with disks of smaller diameter. Furthermore, at most a constant number of disks with larger diameter can be disjoint and intersect a square on the current level. Hence, all such sets of disks can be enumerated in polynomial time for each square, and a dynamic programming approach becomes feasible. The details are given in the next section.

**2. A PTAS for independent set in disk graphs.** Let $k > 1$ be a fixed positive integer. Scale all disks so that the largest disk has diameter 1. Let $d_{\min}$ be the diameter of the smallest disk. Let $\ell = \lfloor \log_{k+1}(1/d_{\min}) \rfloor$. We partition the given set $\mathcal{D}$ of disks into $\ell + 1$ levels. For $0 \leq j \leq \ell$, level $j$ consists of all disks $D_i$ with diameter $d_i$ satisfying $(k+1)^{-j} \geq d_i > (k+1)^{-(j+1)}$. Note that the disk with diameter $d_{\min}$ is on level $\ell$.

An example with three levels is sketched in Figure 2.1.

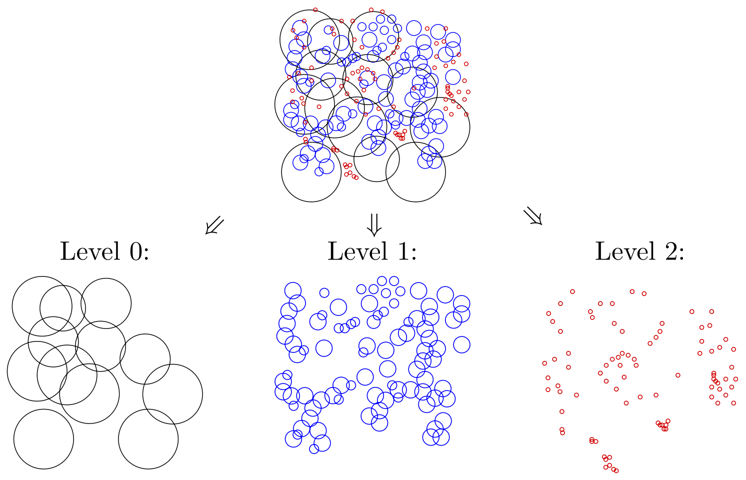


FIG. 2.1. *Partitioning the disks into levels ($k = 2$).*

**2.1. Subdividing the plane.** For each level $j$, $0 \leq j \leq \ell$, we impose a grid on the plane that consists of lines that are $(k+1)^{-j}$ apart from each other. The $v$th vertical line, $v \in \mathbb{Z}$, is at $x = v(k+1)^{-j}$. The $h$th horizontal line, $h \in \mathbb{Z}$, is at $y = h(k+1)^{-j}$. We say that the $v$th vertical line has index $v$ and that the $h$th horizontal line has index $h$. Furthermore, we say that a disk $D_i$ with center $(x_i, y_i)$ and diameter $d_i$ *hits* a vertical line at $x = a$ if $a - d_i/2 < x_i \leq a + d_i/2$. Similarly, we say that $D_i$ hits a horizontal line at $y = b$ if $b - d_i/2 < y_i \leq b + d_i/2$. Intuitively, a disk hits a line if it intersects that line, except if it only touches the line from the left or from below. Note that every disk can hit at most one horizontal line and at most one vertical line on its level.

Let $0 \leq r, s < k$ and consider the vertical lines whose index modulo $k$ equals $r$ and the horizontal lines whose index modulo $k$ equals $s$. We say that these lines are *active* for $(r, s)$. Figure 2.2 illustrates the horizontal grid lines and active lines on two consecutive levels.

Define $\mathcal{D}(r, s)$ to be the set of disks that is obtained from $\mathcal{D}$ by deleting all disks that hit a line that is on the same level as the disk and that is active for $(r, s)$. See Figure 2.3 for an example.

In the following, we write $OPT$ as shorthand for $OPT_{IS}$.

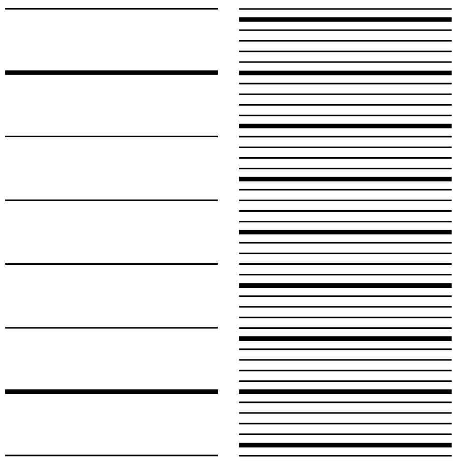


FIG. 2.2. *Horizontal grid lines on level $j$ (left-hand side) and level $j+1$ (right-hand side) for $k=5$. Active lines are drawn bold.*

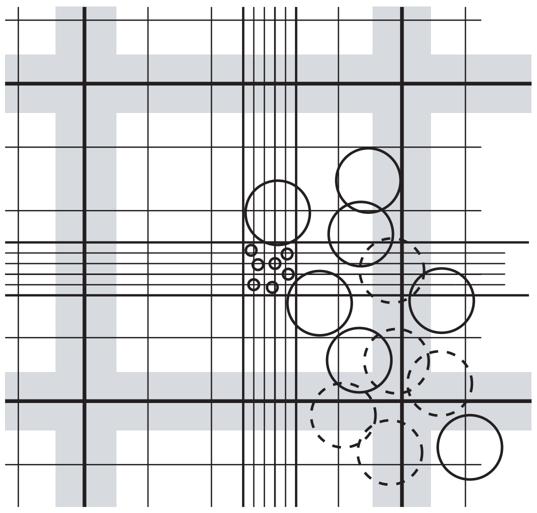


FIG. 2.3. *Example of grid and active lines on level $j$ (coarse grid) and on level $j+1$ (fine grid) for $k=5$. The big disks have level $j$, and the small disks have level $j+1$. All disks shown have the maximum possible diameter on their level. Active lines are drawn bold. Disks that hit active lines are drawn dashed. Note that a disk on level $j$ can hit an active line only if its center is in the shaded strip along that active line.*

LEMMA 2.1. *For at least one pair $(r,s)$, $0 \le r,s < k$, we have $OPT(\mathcal{D}(r,s)) \ge (1-\frac{1}{k})^2 OPT(\mathcal{D})$.*

*Proof.* Let $S^* \subseteq \mathcal{D}$ be any set of disjoint disks with total weight $OPT(\mathcal{D})$.

For $0 \le r < k$, let $S_r^*$ be the set of all disks in $S^*$ that hit a vertical line on their level whose index modulo $k$ is $r$. As the sets $S_r^*$ are disjoint, the weight of at least one of them must be at most a $\frac{1}{k}$-fraction of the weight of $S^*$. For this set $S_r^*$, let $T^* = S^* \setminus S_r^*$ and note that the weight of $T^*$ is at least $(1-\frac{1}{k})OPT(\mathcal{D})$.

For $0 \le s < k$, let $T_s^*$ be the set of all disks in $T^*$ that hit a horizontal line on their level whose index modulo $k$ is $s$. The weight of at least one of these sets $T_s^*$

must be at most a $\frac{1}{k}$-fraction of the weight of $T^*$. For this set $T^*_s$, let $U^* = T^* \setminus T^*_s$. Note that $U^* \subseteq \mathcal{D}(r,s)$ and the weight of $U^*$ is at least $(1-\frac{1}{k})^2 OPT(\mathcal{D})$. ◻

The algorithm considers all $k^2$ possible values for $r$ and $s$ such that $0 \le r,s < k$. For each possibility, an optimal independent set in $\mathcal{D}(r,s)$ is computed using dynamic programming. Among the $k^2$ sets obtained in this way, the one with largest weight is output. By Lemma 2.1, this set has total weight at least $(1-\frac{1}{k})^2 OPT(\mathcal{D})$. Therefore, the algorithm achieves approximation ratio $(1+\frac{1}{k-1})^2$. As $k$ gets larger, the approximation ratio gets arbitrarily close to 1.

It remains to show how an optimal independent set in $\mathcal{D}(r,s)$ can be computed using dynamic programming and that the running-time of the algorithm is polynomial in the size of the input for a fixed value of $k$.

**2.2. Dynamic programming.** Let $0 \le r,s < k$. In this section we discuss the dynamic programming algorithm for computing an optimal independent set in $\mathcal{D}(r,s)$.

Consider one particular level $j$, $0 \le j \le \ell$. The lines on level $j$ that are active for $(r,s)$ partition the plane into squares. More precisely, for consecutive active vertical lines at $x = a_1$ and $x = a_2$ and consecutive active horizontal lines at $y = b_1$ and $y = b_2$, one square $\{(x,y) \mid a_1 < x \le a_2, b_1 < y \le b_2\}$ is obtained. We refer to these squares on level $j$ as *$j$-squares.*

By definition of $\mathcal{D}(r,s)$, every disk in $\mathcal{D}(r,s)$ that is on level $j$ is completely contained in some $j$-square. Furthermore, we have the following lemma about the relationship between squares on different levels.

LEMMA 2.2. *For any $j$, $0 \le j < \ell$, every $(j+1)$-square is completely contained in some $j$-square.*

*Proof.* We prove the lemma by showing that every line that is active for $(r,s)$ on level $j$ is also active for $(r,s)$ on level $j+1$. Figure 2.2 illustrates this claim for the horizontal lines: note that the active lines on level $j$ are active on level $j+1$ also after the active lines are "shifted" up or down on their respective levels.

Without loss of generality, consider only the horizontal lines. Let $y = h(k+1)^{-j}$ be an active horizontal line on level $j$. This means that $h \bmod k = s$. This line is identical to the line $y = h(k+1)(k+1)^{-(j+1)}$, which is on level $j+1$ and has index $h(k+1)$. Obviously, $h(k+1) \bmod k = h \bmod k = s$. Hence, the line is also active on level $j+1$. ◻

COROLLARY 2.3. *Every $j$-square is the union of $(k+1)^2$ $(j+1)$-squares.*

Call a $j$-square $S$ *relevant* if $\mathcal{D}(r,s)$ contains at least one disk of level $j$ that is contained in $S$. For a relevant $j$-square $S$ and a relevant $j'$-square $S'$, $j' > j$, we say that $S'$ is a *child* or *child square* of $S$ (and $S$ is a *parent* of $S'$) if $S'$ is contained in $S$ and if there is no relevant $j''$-square $S''$, $j' > j'' > j$, such that $S'$ is contained in $S''$ and $S''$ is contained in $S$.

The algorithm processes all relevant squares in order of nonincreasing levels. When a $j$-square $S$ is processed, a table $T_S$ is computed. For every set $I$ of disjoint disks of level smaller than $j$ that intersect $S$, the table entry $T_S(I)$ is a maximum weight set of disjoint disks of level at least $j$ that are contained in $S$ and disjoint from the disks in $I$. To formalize this property, we introduce the following definition.

DEFINITION 2.4. *Let $S$ be a relevant $j$-square and let $I$ be a set of disjoint disks of level less than $j$ that intersect $S$. Then the table entry $T_S(I)$ is called* good *if it satisfies the following properties:*

(a) *$T_S(I) \subseteq \mathcal{D}(r,s)$ consists of disks that are contained in $S$ and have level at least $j$.*

(b) *$I \cup T_S(I)$ is an independent set.*

(c) *$w(T_S(I))$ is maximum among all sets that satisfy* (a) *and* (b).

Provided that tables with good entries have been computed for all relevant squares, it is clear that the algorithm can output a maximum weight independent set in $\mathcal{D}(r, s)$ by taking the union of the sets $T_S(\emptyset)$ for all relevant squares $S$ that do not have a parent. In the next section, we give an algorithm to efficiently compute the table $T_S$ for a $j$-square $S$ provided that the tables $T_{S'}$ have already been computed for all child squares $S'$ of $S$.

**2.3. Computing the table for a relevant square.** Consider some relevant $j$-square $S$. First, we give a bound on the number of sets $I$ for which a table entry $T_S(I)$ needs to be computed. For this purpose, we show that the number of disjoint disks of level smaller than $j$ that can intersect a $j$-square is bounded by $O(k^2)$. Figure 2.3 can serve as an illustration of this fact: only $O(k^2)$ disjoint disks whose diameter is larger than that of the big disks can intersect the $j$-square shown in the figure.

LEMMA 2.5. *Let $S$ be some $j$-square and let $I \subseteq \mathcal{D}$ be a set of disjoint disks such that each disk in $I$ has level at most $j - 1$ and intersects $S$. Then there is a constant $C$ such that $|I| \leq Ck^2$.*

*Proof.* Let $\bar{S}$ be a square that consists of $S$ and a strip of width $(k + 1)^{-j}$ surrounding $S$. As the disks in $I$ have level at most $j - 1$, they have diameter larger than $(k + 1)^{-j}$ and, therefore, area larger than $\pi((k + 1)^{-j}/2)^2$. Furthermore, each disk in $I$ occupies an area larger than $\pi((k + 1)^{-j}/2)^2$ within $\bar{S}$: a disk of diameter $(k + 1)^{-j}$ that intersects $S$ would have to be completely contained in $\bar{S}$, and larger disks that intersect $S$ would have to occupy an area of $\bar{S}$ that is even larger. The area of $\bar{S}$ is $((k + 2)(k + 1)^{-j})^2$. Therefore, we must have

$$|I| \leq \frac{((k + 2)(k + 1)^{-j})^2}{\pi((k + 1)^{-j}/2)^2} = \frac{4}{\pi}(k + 2)^2 < 6k^2.$$

Hence, we can choose $C = 6$. ∎

By Lemma 2.5, the algorithm can enumerate all independent sets $I$ of disks that have level smaller than $j$ and that intersect $S$ as follows: It simply enumerates all subsets of at most $Ck^2$ disks of level smaller than $j$ that intersect $S$ and checks for each of them whether it is an independent set. This enumeration can be performed in time $n^{O(k^2)}$.

We consider one such set $I$ and show how $T_S(I)$ is computed. Assume for now that the $j$-square $S$ has either no child square at all or exactly $(k + 1)^2$ child squares on level $j + 1$. Denote the child squares by $S'_{g,h}$, where $g$ is the row index and $h$ is the column index, $0 \leq g, h \leq k$. Thus, $S'_{0,0}$ is the bottom left child square of $S$, and $S'_{k,k}$ is the top right child square. We will show later how to deal with the case that some child squares of $S$ are at a level larger than $j + 1$.

We can assume that $T_{S'_{g,h}}$ has already been computed for each such child square $S'_{g,h}$. A first approach to computing $T_S(I)$, which we have used in an earlier version of our result, is to enumerate all sets $I'$ of disjoint disks of level $j$ that intersect $S$ and that are disjoint from the disks in $I$, and to look up, for each such set $I'$, the table entries $T_{S'_{g,h}}(I'_{g,h})$, where $I'_{g,h}$ is the set of disks in $I \cup I'$ that intersect $S'_{g,h}$. The union of $I'$ and all sets of the form $T_{S'_{g,h}}(I'_{g,h})$ forms a candidate set, and the candidate set of largest weight yields $T_S(I)$. The cardinality of $I'$ can be bounded by $O(k^4)$, and so this approach leads to a running-time of $n^{O(k^4)}$. In the following, we present an improved algorithm based on dynamic programming that allows us to reduce the running-time to $n^{O(k^2)}$.

By $S'_{g_1\cdot\cdot g_2,h_1\cdot\cdot h_2}$, where $0 \le g_1 \le g_2 \le k$ and $0 \le h_1 \le h_2 \le k$, we denote the union of all child squares $S'_{g,h}$ with $g_1 \le g \le g_2$ and $h_1 \le h \le h_2$. We call such a union of child squares a *rectangle*. We say that a disk $D$ *intersects the boundary* of a rectangle $R$ if $D \cap R$ and $D \setminus R$ are both nonempty.

In order to determine $T_S(I)$, the algorithm computes an auxiliary table $AT_{S,I}$ with entries $AT_{S,I}(S'_{g_1\cdot\cdot g_2,h_1\cdot\cdot h_2}, J)$ for certain values of $g_1, g_2, h_1, h_2$, where $J$ is a set of disks of level $j$ that intersect the boundary of $S'_{g_1\cdot\cdot g_2,h_1\cdot\cdot h_2}$ and have the property that $I \cup J$ is an independent set. The table entry $AT_{S,I}(S'_{g_1\cdot\cdot g_2,h_1\cdot\cdot h_2}, J)$ is a maximum weight set $J'$ of disks that have level at least $j$, are contained in $S'_{g_1\cdot\cdot g_2,h_1\cdot\cdot h_2}$, and have the property that $I \cup J \cup J'$ is an independent set. This property is captured formally in the following definition.

DEFINITION 2.6. *Let $S$ be a relevant $j$-square and let $I$ be a set of disjoint disks of level less than $j$ that intersect $S$. Let $S'_{g_1\cdot\cdot g_2,h_1\cdot\cdot h_2}$ be a rectangle of child squares of $S$, and let $J$ be a set of disks of level $j$ that intersect the boundary of $S'_{g_1\cdot\cdot g_2,h_1\cdot\cdot h_2}$ and have the property that $I \cup J$ is an independent set.*

*Then the table entry $AT_{S,I}(S'_{g_1\cdot\cdot g_2,h_1\cdot\cdot h_2}, J)$ is called* good *if it satisfies the following properties:*

(a) *$AT_{S,I}(S'_{g_1\cdot\cdot g_2,h_1\cdot\cdot h_2}, J) \subseteq \mathcal{D}(r,s)$ consists of disks that are contained in $S'_{g_1\cdot\cdot g_2,h_1\cdot\cdot h_2}$ and have level at least $j$.*

(b) *$I \cup J \cup AT_{S,I}(S'_{g_1\cdot\cdot g_2,h_1\cdot\cdot h_2}, J)$ is an independent set.*

(c) *$w(AT_{S,I}(S'_{g_1\cdot\cdot g_2,h_1\cdot\cdot h_2}, J))$ is maximum among all sets that satisfy* (a) *and* (b).

Once a good table entry $AT_{S,I}(S'_{0\cdot\cdot k,0\cdot\cdot k}, \emptyset)$ is computed, it immediately yields $T_S(I)$.

Table entries $AT_{S,I}(S'_{g_1\cdot\cdot g_2,h_1\cdot\cdot h_2}, J)$ are again computed by dynamic programming. First, we bound the number of different sets $J$ that must be considered as table index for a rectangle $S'_{g_1\cdot\cdot g_2,h_1\cdot\cdot h_2}$.

LEMMA 2.7. *Let $S'_{g_1\cdot\cdot g_2,h_1\cdot\cdot h_2}$ be a rectangle and let $J$ be a set of disjoint disks of level $j$ such that all disks in $J$ intersect the boundary of $S'_{g_1\cdot\cdot g_2,h_1\cdot\cdot h_2}$. Then there is a constant $C'$ such that $|J| \le C'k^2$.*

*Proof.* Let $R = S'_{g_1\cdot\cdot g_2,h_1\cdot\cdot h_2}$. The boundary $B$ of $R$ consists of line segments with total length at most $4k(k+1)^{-j}$. Let $\bar{B}$ be obtained by extending $B$ by a strip of width $(k+1)^{-(j+1)}$ on the inside of $B$ and on the outside of $B$. The total area of $\bar{B}$ is at most $8k(k+1)^{-2j-1}$. Each disk of level $j$ that intersects $B$ occupies an area at least $\pi(k+1)^{-2(j+1)}/4$ of $\bar{B}$. Therefore, $J$ can contain at most

$$\frac{8k(k+1)^{-2j-1}}{\frac{\pi}{4}(k+1)^{-2(j+1)}} = \frac{32}{\pi}k(k+1) \le 16k^2$$

disjoint disks. So we can take $C' = 16$. ∎

Lemma 2.7 shows that for each rectangle $S'_{g_1\cdot\cdot g_2,h_1\cdot\cdot h_2}$, there are at most $n^{O(k^2)}$ different sets $J$ that need to be considered as table index with respect to $AT_{S,I}$ $(S'_{g_1\cdot\cdot g_2,h_1\cdot\cdot h_2}, J)$.

Here and in the following, we use $AT_{S,I}(S'_{g_1\cdot\cdot g_2,h_1\cdot\cdot h_2}, *)$ as a shorthand notation to refer to the table entries $AT_{S,I}(S'_{g_1\cdot\cdot g_2,h_1\cdot\cdot h_2}, J)$ for all relevant values of $J$.

As the basis of the dynamic programming, the table entries $AT_{S,I}(S'_{g\cdot\cdot g,h\cdot\cdot h}, *)$ are computed for $0 \le g, h \le k$ as shown in Figure 2.4. Note that $S'_{g\cdot\cdot g,h\cdot\cdot h} = S'_{g,h}$. For each child square $S'_{g,h}$, all independent sets $U$ of disks of level $j$ intersecting $S'_{g,h}$ are enumerated. This can be done in time $n^{O(k^2)}$ by Lemma 2.5. If $I \cup U$ is

```
**Input:** square $S$ on level $j$,
           set $I$ of disjoint disks of level $< j$ intersecting $S$,
           integers $g, h$ with $0 \le g, h \le k$
**Output:** table entries $AT_{S,I}(S'_{g,h}, J)$ for all $J$
$AT_{S,I}(S'_{g,h}, *) \leftarrow$ undefined;
$Q \leftarrow$ all disks in $\mathcal{D}(r,s)$ of level $j$ intersecting $S'_{g,h}$;
**for** all $U \subseteq Q$ such that $|U| \le Ck^2$ **do**
    **if** the disks in $I \cup U$ are disjoint **then**
        $I' \leftarrow \{D \in I \mid D$ intersects $S'_{g,h}\}$;
        $X \leftarrow T_{S'_{g,h}}(I' \cup U)$;
        $X \leftarrow X \cup \{D \in U \mid D$ is contained in $S'_{g,h}\}$;
        $J \leftarrow \{D \in U \mid D$ intersects the boundary of $S'_{g,h}\}$;
        **if** $AT_{S,I}(S'_{g,h}, J)$ is undefined **or**
           $w(X) > w(AT_{S,I}(S'_{g,h}, J))$ **then**
              $AT_{S,I}(S'_{g,h}, J) \leftarrow X$;
        **fi**
    **fi**
**od**
```

FIG. 2.4. *Computing the auxiliary table $AT_{S,I}(S'_{g,h}, *)$.*

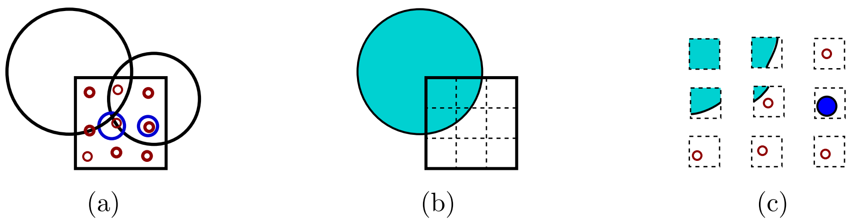


FIG. 2.5. *Example of table lookups for a square $S$ at level $j$ in case $k = 2$.* (a) *shows* 13 *disks in $\mathcal{D}(r,s)$ that intersect $S$:* 2 *disks of level less than $j$,* 2 *disks on level $j$, and* 9 *disks on level $j+1$.* (b) *displays an independent set $I$ consisting of* 1 *disk of level less than $j$.* (c) *illustrates that lookups are performed in* 9 *tables $T_{S'_{g,h}}$ during the computation of the table entries $AT_{S,I}(S'_{g,h}, *)$.*

an independent set, the optimal way of extending the set $I \cup U$ to a larger weight independent set by adding disks of larger levels that are contained in $S'_{g,h}$ is computed by looking up $T_{S'_{g,h}}(I' \cup U)$, where $I' = \{D \in I \mid D$ intersects $S'_{g,h}\}$. (If $S$ does not have any relevant child squares, all lookups in tables $T_{S'_{g,h}}$ are taken to return the empty set.) Let $J = \{D \in U \mid D$ intersects the boundary of $S'_{g,h}\}$. If the independent set obtained in this way has larger weight than the previous set stored in table entry $AT_{S,I}(S'_{g,h}, J)$, the entry is updated to store the new set. An example of the table lookups performed for a relevant $j$-square $S$ in the tables $T_{S'_{g,h}}$ of subsquares at level $j+1$ is sketched in Figure 2.5.

Next, we show how to combine the information from the table entries of two rectangles to obtain the table entries for the rectangle representing the union of the two rectangles. Without loss of generality, we discuss only the case where the two rectangles share a horizontal edge and have the same width. The combination of rectangles that share a vertical edge and have the same height is analogous. It is clear that $(k+1)^2-1$ such combinations suffice to obtain the table entry $AT_{S,I}(S'_{0\cdot\cdot k,0\cdot\cdot k}, \emptyset)$,

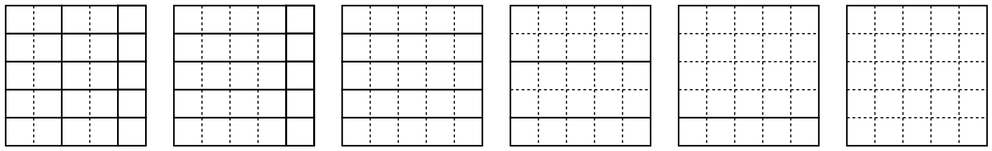


FIG. 2.6. *Combining subsquares into rectangles.*

```
Input: square $S$ on level $j$,
        set $I$ of disjoint disks of level $< j$ intersecting $S$,
        integers $g_1, g_2, g_3$ with $0 \le g_1 \le g_2 < g_3 \le k$,
        integers $h_1, h_2$ with $0 \le h_1 \le h_2 \le k$,
        previously computed table entries $AT_{S,I}(S'_{g_1 \cdot\cdot g_2, h_1 \cdot\cdot h_2}, *)$
            and $AT_{S,I}(S'_{g_2+1 \cdot\cdot g_3, h_1 \cdot\cdot h_2}, *)$
Output: table entries $AT_{S,I}(S'_{g_1 \cdot\cdot g_3, h_1 \cdot\cdot h_2}, J)$ for all $J$
$R_1 \leftarrow S'_{g_1 \cdot\cdot g_2, h_1 \cdot\cdot h_2}$;
$R_2 \leftarrow S'_{g_2+1 \cdot\cdot g_3, h_1 \cdot\cdot h_2}$;
$AT_{S,I}(S'_{g_1 \cdot\cdot g_3, h_1 \cdot\cdot h_2}, *) \leftarrow$ undefined;
$Q \leftarrow$ all disks in $\mathcal{D}(r, s)$ of level $j$ intersecting the boundary of $R_1$ or $R_2$;
for all $U \subseteq Q$ such that $|U| \le 2C'k^2$ do
    if the disks in $I \cup U$ are disjoint then
        for $i = 1$ to 2 do
            $U_i \leftarrow \{D \in U \mid D$ intersects the boundary of $R_i\}$;
            $X_i \leftarrow AT_{S,I}(R_i, U_i)$;
        od;
        $X \leftarrow X_1 \cup X_2 \cup \{D \in U \mid D$ does not intersect the boundary of
                $S'_{g_1 \cdot\cdot g_3, h_1 \cdot\cdot h_2}\}$;
        $J \leftarrow \{D \in U \mid D$ intersects the boundary of $S'_{g_1 \cdot\cdot g_3, h_1 \cdot\cdot h_2}\}$;
        if $AT_{S,I}(S'_{g_1 \cdot\cdot g_3, h_1 \cdot\cdot h_2}, J)$ is undefined or
          $w(X) > w(AT_{S,I}(S'_{g_1 \cdot\cdot g_3, h_1 \cdot\cdot h_2}, J)$ then
            $AT_{S,I}(S'_{g_1 \cdot\cdot g_3, h_1 \cdot\cdot h_2}, J) \leftarrow X$;
        fi
    fi
od
```

FIG. 2.7. *Computing the auxiliary table* $AT_{S,I}(S'_{g_1 \cdot\cdot g_3, h_1 \cdot\cdot h_2}, *)$.

which then gives $T_S(I)$. See Figure 2.6 for an example with $k = 4$ in which first the horizontally adjacent rectangles within each row are combined and then the vertically adjacent row rectangles are combined.

The algorithm for computing the table entries $AT_{S,I}(S'_{g_1 \cdot\cdot g_3, h_1 \cdot\cdot h_2}, *)$ from the table entries $AT_{S,I}(S'_{g_1 \cdot\cdot g_2, h_1 \cdot\cdot h_2}, *)$ and $AT_{S,I}(S'_{g_2+1 \cdot\cdot g_3, h_1 \cdot\cdot h_2}, *)$ for some $g_1 \le g_2 < g_3, h_1 \le h_2$ is shown in Figure 2.7. Let $R_1 = S'_{g_1 \cdot\cdot g_2, h_1 \cdot\cdot h_2}$ and $R_2 = S'_{g_2+1 \cdot\cdot g_3, h_1 \cdot\cdot h_2}$. The algorithm enumerates all independent sets $U$ of disks of level $j$ that intersect the boundary of $R_1$ or $R_2$. As each such set has cardinality at most $2C'k^2$, where $C'$ is the constant from Lemma 2.7, there are at most $n^{O(k^2)}$ such sets. If $I \cup U$ is an independent set, the optimal way of extending the set to a larger weight independent set by adding disks of level at least $j$ that are contained in $R_1$ or $R_2$ is computed by looking up $AT_{S,I}(R_1, U_1)$ and $AT_{S,I}(R_2, U_2)$, where $U_i$ for $i = 1, 2$ is the set of disks in $U$ that intersect the boundary of $R_i$. Then the table entry $AT_{S,I}(S'_{g_1 \cdot\cdot g_3, h_1 \cdot\cdot h_2}, J)$ is updated for the appropriate choice of $J$ provided that the new set is better than the previously stored entry. When all sets $U$ have been processed in this way, the table entries $AT_{S,I}(S'_{g_1 \cdot\cdot g_3, h_1 \cdot\cdot h_2}, *)$ have their final values.

So far we have assumed that the current $j$-square $S$ has either no child squares or exactly $(k+1)^2$ child squares on level $j+1$. It is easy to handle the case that $S$ has fewer than $(k+1)^2$ child squares on level $j+1$ and possibly some child squares on levels larger than $j+1$: Just before the computation of $T_S$, we compute the tables $T_{S'_{g,h}}$ for all $(k+1)^2$ $(j+1)$-squares $S'_{g,h}$ contained in $S$. For the $(j+1)$-squares that are relevant, the tables are already computed. For every $(j+1)$-square $S'_{g,h}$ that is not relevant, we can enumerate all $n^{O(k^2)}$ sets $I$ of disjoint disks of level at most $j$ that intersect $S'_{g,h}$ and compute $T_{S'_{g,h}}(I)$ by taking the union of the sets $T_{S''}(I^{S''})$ for all child squares $S''$ of $S$ that are contained in $S'_{g,h}$. Here, $I^{S''} = \{D \in I \mid D \text{ intersects } S''\}$. It is clear that the table entries $T_{S'_{g,h}}(I)$ computed in this way are good provided that good entries of the tables $T_{S''}$ have been computed previously.

This completes the description of the algorithm. In summary, the optimal independent set in $\mathcal{D}(r,s)$ is computed by dynamic programming on the relevant squares using table entries $T_S(I)$, while each such entry is computed by dynamic programming on rectangles within the current square using auxiliary table entries $AT_{S,I}(S'_{g_1 \cdot\cdot g_2, h_1 \cdot\cdot h_2}, J)$.

## 2.4. Running-time of the algorithm.

LEMMA 2.8. *The running-time of the algorithm is $n^{O(k^2)}$ and hence polynomial for any fixed $k > 1$.*

*Proof.* There are $k^2$ sets $\mathcal{D}(r,s)$ that have to be considered. As there are at most $n$ disks in $\mathcal{D}(r,s)$, there can be at most $n$ relevant squares. The relevant squares and their forest structure (the links between every relevant square and its children) can be computed easily in time polynomial in $n$.

For each relevant square $S$, the algorithm first computes the missing tables $T_{S'_{g,h}}$ for $(j+1)$-squares $S'_{g,h}$ that are contained in $S$, as discussed at the end of section 2.3. Each of the $O(k^2)$ such $(j+1)$-squares can be handled in time $n^{O(k^2)}$. Then the algorithm enumerates $n^{O(k^2)}$ sets $I$ for which a table entry $T_S(I)$ has to be computed. Each such entry is determined by using dynamic programming to fill the table $AT_{S,I}$. Table entries $AT_{S,I}(S'_{g_1 \cdot\cdot g_2, h_1 \cdot\cdot h_2}, J)$ are computed for $O(k^2)$ different rectangles $S'_{g_1 \cdot\cdot g_2, h_1 \cdot\cdot h_2}$, and for each rectangle the computation takes time $n^{O(k^2)}$, as can be seen from the pseudocode in Figures 2.4 and 2.7.

Thus, the total running-time can be bounded by

$$k^2 \cdot n^{O(1)} \cdot \left( O(k^2) \cdot n^{O(k^2)} + n^{O(k^2)} \cdot O(k^2) \cdot n^{O(k^2)} \right) = n^{O(k^2)}. \qquad \square$$

## 2.5. Correctness of the algorithm.

LEMMA 2.9. *The entries of all the tables $T_S$ and $AT_{S,I}$ computed by the algorithm are good.*

*Proof.* First, it is clear that the table entries computed by the algorithm satisfy properties (a) and (b) of Definitions 2.4 and 2.6. It remains to show that the sets stored in the tables are indeed of maximum weight.

The proof is by induction on the number of squares processed by the algorithm. Initially, the claim of the lemma holds vacuously. Now assume that the algorithm processes a relevant $j$-square $S$ and that good entries for the tables $T_{S'}$ have been computed for all relevant squares $S'$ that were processed before $S$. Let $I$ be a set of disjoint disks of level smaller than $j$ that intersect $S$.

Consider the computation of the table entries $AT_{S,I}(S'_{g,h}, J)$ as shown in Figure 2.4. Fix some set $J$ of disks at level $j$ intersecting the boundary of $S'_{g,h}$ such that $I \cup J$ is an independent set. Let $X^*$ be a maximum weight set of disks at level at least $j$ that are contained in $S'_{g,h}$ such that $I \cup J \cup X^*$ is an independent set. Let $X^*_{=j}$ be the set of disks at level $j$ in $X^*$ and let $U^* = J \cup X^*_{=j}$. Then $U^*$ is one of the sets $U$ enumerated by the algorithm, and we consider the iteration of the for-loop when this set is processed. Since the entries of $T_{S'_{g,h}}$ are good, the lookup in $T_{S'_{g,h}}(I' \cup U)$ returns a set of weight at least $w(X^* \setminus X^*_{=j})$. Then the set

$$\begin{aligned} X &= T_{S'_{g,h}}(I' \cup U) \cup \{D \in U \mid D \text{ is contained in } S'_{g,h}\} \\ &= T_{S'_{g,h}}(I' \cup U) \cup X^*_{=j} \end{aligned}$$

computed by the algorithm has weight at least $w(X^* \setminus X^*_{=j}) + w(X^*_{=j}) = w(X^*)$. Hence, the table entry $AT_{S,I}(S'_{g,h}, J)$ contains a set $X$ of weight at least $w(X^*)$ when the algorithm of Figure 2.4 terminates. Since $AT_{S,I}(S'_{g,h}, J)$ satisfies properties (a) and (b) of Definition 2.6 and $X^*$ is a maximum weight set with this property, we get that $w(AT_{S,I}(S'_{g,h}, J)) = w(X^*)$. Hence, the computed table entry $AT_{S,I}(S'_{g,h}, J)$ is good.

Consider the computation of a table entry $AT_{S,I}(S'_{g_1\cdot\cdot g_3, h_1\cdot\cdot h_2}, J)$ as shown in Figure 2.7. Assume that the previously computed entries $AT_{S,I}(S'_{g_1\cdot\cdot g_2, h_1\cdot\cdot h_2}, *)$ and $AT_{S,I}(S'_{g_2+1\cdot\cdot g_3, h_1\cdot\cdot h_2}, *)$ are good. Let $R_1 = S'_{g_1\cdot\cdot g_2, h_1\cdot\cdot h_2}$ and $R_2 = S'_{g_2+1\cdot\cdot g_3, h_1\cdot\cdot h_2}$.

Fix some set $J$ of disks at level $j$ intersecting the boundary of $S'_{g_1\cdot\cdot g_3, h_1\cdot\cdot h_2} = R_1 \cup R_2$ such that $I \cup J$ is an independent set. Let $X^*$ be a maximum weight set of disks at level at least $j$ that are contained in $R_1 \cup R_2$ such that $I \cup J \cup X^*$ is an independent set. Let $X^*_{1,2}$ be the set of disks at level $j$ in $X^*$ that intersect the boundary of $R_1$ and the boundary of $R_2$. Let $X^*_1$ and $X^*_2$ be the set of disks in $X^*$ that are contained in $R_1$ and in $R_2$, respectively.

Let $U^* = J \cup X^*_{1,2}$. Then $U^*$ is one of the sets $U$ enumerated by the algorithm. For this set $U^*$, the table lookups $AT_{S,I}(R_i, U_i)$ for $i = 1, 2$, with $U_i$ calculated as shown in Figure 2.7, yield disjoint sets $X_1$ and $X_2$ of weight at least $w(X^*_1)$ and $w(X^*_2)$, respectively. Then the set

$$\begin{aligned} X &= X_1 \cup X_2 \cup \{D \in U \mid D \text{ does not intersect the boundary of } S'_{g_1\cdot\cdot g_3, h_1\cdot\cdot h_2}\} \\ &= X_1 \cup X_2 \cup X^*_{1,2} \end{aligned}$$

calculated by the algorithm has weight at least $w(X^*_1) + w(X^*_2) + w(X^*_{1,2}) = w(X^*)$. Hence, the table entry $AT_{S,I}(R_1 \cup R_2, J)$ contains a set $X$ of weight at least $w(X^*)$ when the algorithm of Figure 2.7 terminates, and is thus a good entry.

So we see that the computed auxiliary table entries $AT_{S,I}(S'_{g_1\cdot\cdot g_2, h_1\cdot\cdot h_2}, *)$ are indeed good for the rectangles $S'_{g_1\cdot\cdot g_2, h_1\cdot\cdot h_2}$. Since the algorithm then sets $T_S(I)$ equal to $AT_{S,I}(S, \emptyset)$, this shows that the computed entries of $T_S(I)$ are good as well. $\square$

We observe that all disks in $\mathcal{D}(r, s)$ are contained in some relevant square without parent, and that all relevant squares without parent are disjoint. By Lemma 2.9, the computed table entries $T_S(\emptyset)$ are good for all relevant squares without parent, and this shows that the algorithm indeed computes an optimal independent set in $\mathcal{D}(r, s)$. Together with the discussion in section 2.1, we have that the algorithm achieves approximation ratio $(1 + \frac{1}{k-1})^2 \leq 1 + \frac{3}{k-1}$. In order to achieve approximation ratio

$1+\varepsilon$, we can set $k = \lceil 3/\varepsilon \rceil + 1$. The running-time is $n^{O(k^2)}$ and hence polynomial for any fixed $k > 1$. So the algorithm indeed constitutes a PTAS. We summarize our result in the following theorem.

THEOREM 2.10. *There is a PTAS for MWIS in disk graphs, provided that a disk representation of the graph is given. The running-time for achieving approximation ratio* $1+\varepsilon$ *is* $n^{O(1/\varepsilon^2)}$ *for a disk graph with* $n$ *disks.*

## 3. A PTAS for vertex cover in disk graphs.

In this section we describe a PTAS for MWVC. The basic approach is similar to the PTAS for MWIS of the previous section, but a number of details cause additional technical difficulties and require a different treatment. One problem is that the size of a vertex cover cannot be bounded by area arguments as the size of an independent set can. The main idea to circumvent this problem is to work with the complements of vertex covers, which are independent sets.

The partitioning of the disks into levels and the subdivision of the plane into squares at each level is the same as for MWIS. Again, all values of $r$ and $s$ such that $0 \leq r, s < k$ are considered in turn. Note that the definition of the squares on each level depends on $r$ and $s$. Contrary to the PTAS for MWIS, disks that are not completely contained in some square on their level are not removed; instead, these disks are considered in all squares on their level that they intersect (there are at most four such squares). Now a $j$-square $S$ is called relevant if $\mathcal{D}$ contains a disk of level $j$ that intersects $S$.

Consider some $j$-square $S$ and let $\mathcal{D}^S$ denote the set of all disks in $\mathcal{D}$ that intersect $S$. Let $\mathcal{D}^S_{<j}$ be the set of disks in $\mathcal{D}^S$ that have level smaller than $j$. Define $\mathcal{D}^S_{\leq j}$, $\mathcal{D}^S_{=j}$, $\mathcal{D}^S_{\geq j}$, and $\mathcal{D}^S_{>j}$ analogously.

We say that a set $Z \subseteq \mathcal{D}$ is a *pseudocover* of $S$ if, for any two disks $D_1, D_2 \in \mathcal{D}^S$ that intersect in $S$ (i.e., $D_1 \cap D_2 \cap S \neq \emptyset$), $Z$ contains $D_1$ or $D_2$ (or both). Note that a pseudocover of $S$ need not contain $D_1$ or $D_2$ if the only intersection of $D_1$ with $D_2$ is outside $S$. We observe that, for any two disks in $\mathcal{D}$ that intersect, there exists a relevant square in which they intersect.

For any pseudocover $Z$ of $S$, call $\mathcal{D}^S_{<j} \cap Z$ the *projection* of $Z$ onto $\mathcal{D}^S_{<j}$. While processing $S$, the algorithm considers only pseudocovers $Z$ of $S$ for which the projection of $Z$ onto $\mathcal{D}^S_{<j}$ equals $\mathcal{D}^S_{<j} \setminus I$ for some independent set $I \subseteq \mathcal{D}^S_{<j}$. As the number of independent sets in $\mathcal{D}^S_{<j}$ is bounded by $n^{O(k^2)}$ by Lemma 2.5, we can enumerate all of them and thus, by taking the complements, also the corresponding projections of pseudocovers of $S$ onto $\mathcal{D}^S_{<j}$ in time $n^{O(k^2)}$.

As in the PTAS for MWIS, all relevant squares are processed in a bottom-up fashion, and for each square $S$ a table $T_S$ is computed. For a relevant $j$-square $S$ and a set $I$ of disjoint disks in $\mathcal{D}^S_{<j}$, the table entry $T_S(I)$ has the following property.

PROPERTY 1. *For every relevant $j$-square $S$ and every set $I$ of disjoint disks in $\mathcal{D}^S_{<j}$, the table entry $T_S(I)$ is a subset of $\mathcal{D}^S_{\geq j}$ such that the set*

$$(\mathcal{D}^S_{<j} \setminus I) \cup T_S(I)$$

*is a pseudocover of $S$.*

Unlike in the PTAS for MWIS, the entries $T_S(I)$ will not be optimal (minimum weight) sets with the stated property, but will still be good enough to achieve the desired approximation ratio. It is clear that we cannot expect to compute minimum weight entries $T_S(I)$, as in that case the entry $T_S(\emptyset)$ of a relevant square $S$ without parent would represent an optimal vertex cover of all disks contained in $S$, thus solving an $\mathcal{NP}$-hard problem optimally. (Note that we do not delete any disks from the given instance of MWVC in disk graphs before we compute the table entries, contrary to our algorithm for MWIS.)

In the end, the algorithm outputs the union of the sets $T_S(\emptyset)$ for all relevant squares $S$ without parent. By the definition of pseudocovers, this union is a vertex cover of $\mathcal{D}$.

At a relevant $j$-square $S$, all independent sets $I$ in $\mathcal{D}^S_{<j}$ are enumerated in time $n^{O(k^2)}$. The computation of the table entry $T_S(I)$ for one such set $I$ is described in the following.

Assume again that the $j$-square $S$ has $(k+1)^2$ child squares on level $j+1$ (with the same justification as in the case of MWIS) and denote these child squares by $S'_{g,h}$, where $g$ is the row index and $h$ is the column index, $0 \le g, h \le k$. Since the algorithm processes the relevant squares in order of nonincreasing levels, the table $T_{S'_{g,h}}$ has already been computed for each such child square $S'_{g,h}$.

In order to determine $T_S(I)$, the algorithm computes an auxiliary table $AT_{S,I}$ with entries $AT_{S,I}(S'_{g_1 \cdot\cdot g_2, h_1 \cdot\cdot h_2}, J)$ for certain values of $g_1, g_2, h_1, h_2$, where $J$ is a set of disks that are on level $j$ and intersect the boundary of $S'_{g_1 \cdot\cdot g_2, h_1 \cdot\cdot h_2}$ such that $I \cup J$ is an independent set. For a rectangle $R$, let $\mathcal{D}^{\partial R}$ be the set of all disks in $\mathcal{D}$ that intersect the boundary of $R$.

PROPERTY 2. *Let $S$ be a relevant $j$-square and $I$ an independent set in $\mathcal{D}^S_{<j}$. Let $R = S'_{g_1 \cdot\cdot g_2, h_1 \cdot\cdot h_2}$ be a rectangle of child squares of $S$ and $J$ a set of disks of level $j$ intersecting the boundary of $R$ such that $I \cup J$ is an independent set. The table entry $AT_{S,I}(R, J)$ is a set of disks in $\mathcal{D}$ that*

- *either have level $j$ and are contained in $R$ or*
- *have level at least $j+1$ and intersect $R$*

*such that*

$$(\mathcal{D}^S_{<j} \setminus I) \cup (\mathcal{D}^{\partial R}_{=j} \setminus J) \cup AT_{S,I}(R, J)$$

*is a pseudocover of $R$.*

Once the table entries $AT_{S,I}(S'_{0 \cdot\cdot k, 0 \cdot\cdot k}, J) = AT_{S,I}(S, J)$ for all $J$ are computed, $T_S(I)$ is obtained by taking the set $AT_{S,I}(S, J) \cup (\mathcal{D}^{\partial S}_{=j} \setminus J)$ of minimum weight (over all $J$).

The table entries $AT_{S,I}(S'_{g_1 \cdot\cdot g_2, h_1 \cdot\cdot h_2}, J)$ are again computed by dynamic programming. By Lemma 2.7, for each rectangle $S'_{g_1 \cdot\cdot g_2, h_1 \cdot\cdot h_2}$ there are at most $n^{O(k^2)}$ different sets that are relevant as table index $J$ for $AT_{S,I}(S'_{g_1 \cdot\cdot g_2, h_1 \cdot\cdot h_2}, J)$.

As the basis of the dynamic programming, the table entries $AT_{S,I}(S'_{g \cdot\cdot g, h \cdot\cdot h}, J)$ are computed for $0 \le g, h \le k$ as shown in Figure 3.1. For each child square $S'_{g,h}$, all independent sets $U$ of disks of level $j$ intersecting $S'_{g,h}$ are enumerated in time $n^{O(k^2)}$ (by Lemma 2.5). If $I \cup U$ is an independent set, the table entry $T_{S'_{g,h}}(I' \cup U)$ is looked up, where $I' = \{D \in I \mid D \text{ intersects } S'_{g,h}\}$, in order to obtain the pseudocover of $S'_{g,h}$ given by the set

$$(\mathcal{D}^S_{<j} \setminus I) \cup (\mathcal{D}^{S'_{g,h}}_{=j} \setminus U) \cup T_{S'_{g,h}}(I' \cup U).$$

Then $J$ is taken to be the set of disks in $U$ intersecting the boundary of $S'_{g,h}$, and the table entry $AT_{S,I}(S'_{g,h}, J)$ is updated appropriately if the current pseudocover has smaller weight than the one previously stored. (Of course, if $S$ does not have any child squares, the table lookups $T_{S'_{g,h}}(I' \cup U)$ are taken to return the empty set.)

```
Input: square $S$ on level $j$,
       set $I$ of disjoint disks of level $< j$ intersecting $S$,
       integers $g, h$ with $0 \leq g, h \leq k$
Output: table entries $AT_{S,I}(S'_{g,h}, J)$ for all $J$
$AT_{S,I}(S'_{g,h}, *) \leftarrow$ undefined;
$Q \leftarrow$ all disks in $\mathcal{D}$ of level $j$ intersecting $S'_{g,h}$;
for all $U \subseteq Q$ such that $|U| \leq Ck^2$ do
    if the disks in $I \cup U$ are disjoint then
        $I' \leftarrow \{D \in I \mid D$ intersects $S'_{g,h}\}$;
        $X \leftarrow T_{S'_{g,h}}(I' \cup U)$;
        $X \leftarrow X \cup \{D \in \mathcal{D}_{=j}^{S'_{g,h}} \mid D$ is contained in $S'_{g,h}$ and $D \notin U\}$;
        $J \leftarrow \{D \in U \mid D$ intersects the boundary of $S'_{g,h}\}$;
        if $AT_{S,I}(S'_{g,h}, J)$ is undefined or
          $w(X) < w(AT_{S,I}(S'_{g,h}, J)$ then
            $AT_{S,I}(S'_{g,h}, J) \leftarrow X$;
        fi
    fi
od
```

FIG. 3.1. *Computing the auxiliary table $AT_{S,I}(S'_{g,h}, *)$ for MWVC.*

Next, the information from the tables of two rectangles is combined to obtain the table for the rectangle representing the union of the two rectangles. Again, we discuss only the case where the two rectangles share a horizontal edge and have the same width. The algorithm for computing the table $AT_{S,I}(S'_{g_1 \cdot\cdot g_3, h_1 \cdot\cdot h_2}, *)$ from the tables $AT_{S,I}(S'_{g_1 \cdot\cdot g_2, h_1 \cdot\cdot h_2}, *)$ and $AT_{S,I}(S'_{g_2+1 \cdot\cdot g_3, h_1 \cdot\cdot h_2}, *)$ for some $g_1 \leq g_2 < g_3, h_1 \leq h_2$ is shown in Figure 3.2. Let $R_1 = S'_{g_1 \cdot\cdot g_2, h_1 \cdot\cdot h_2}$ and $R_2 = S'_{g_2+1 \cdot\cdot g_3, h_1 \cdot\cdot h_2}$. The algorithm enumerates all independent sets $U$ of disks of level $j$ that intersect the boundary of $R_1$ or $R_2$. As each such set has cardinality at most $2C'k^2$, where $C'$ is the constant from Lemma 2.7, there are at most $n^{O(k^2)}$ such sets. If $I \cup U$ is an independent set, the table entries $X_1 = AT_{S,I}(R_1, U_1)$ and $X_2 = AT_{S,I}(R_2, U_2)$, where $U_1$ and $U_2$ are calculated as shown in Figure 3.2, are looked up to obtain the pseudocover of $R_1 \cup R_2 = S'_{g_1 \cdot\cdot g_3, h_1 \cdot\cdot h_2}$ given by the set

$$(\mathcal{D}_{<j}^{S} \setminus I) \cup ((\mathcal{D}_{=j}^{\partial R_1} \cup \mathcal{D}_{=j}^{\partial R_2}) \setminus U) \cup X_1 \cup X_2.$$

Then the table entry $AT_{S,I}(S'_{g_1 \cdot\cdot g_3, h_1 \cdot\cdot h_2}, J)$ for

$$J = \{D \in U \mid D \text{ intersects the boundary of } S'_{g_1 \cdot\cdot g_3, h_1 \cdot\cdot h_2}\}$$

is updated appropriately if this pseudocover has smaller weight than the one previously stored.

In the end, the algorithm outputs the union of the sets $T_S(\emptyset)$, taken over all relevant squares $S$ that do not have a parent. We will see that this is a $(1 + \frac{6}{k})$-approximation of the minimum weight vertex cover.

### 3.1. The algorithm outputs a vertex cover.

LEMMA 3.1. *The algorithm outputs a vertex cover of $\mathcal{D}$.*

*Proof.* We prove by induction on the number of processed squares that Properties 1 and 2 hold for all computed table entries. Let $S$ be some $j$-square. Recall that

```
Input: square $S$ on level $j$,
        set $I$ of disjoint disks of level $< j$ intersecting $S$,
        integers $g_1, g_2, g_3$ with $0 \leq g_1 \leq g_2 < g_3 \leq k$,
        integers $h_1, h_2$ with $0 \leq h_1 \leq h_2 \leq k$,
        previously computed table entries $AT_{S,I}(S'_{g_1 \cdot\cdot g_2, h_1 \cdot\cdot h_2}, *)$ and
            $AT_{S,I}(S'_{g_2+1 \cdot\cdot g_3, h_1 \cdot\cdot h_2}, *)$
Output: table entries $AT_{S,I}(S'_{g_1 \cdot\cdot g_3, h_1 \cdot\cdot h_2}, J)$ for all $J$
$R_1 \leftarrow S'_{g_1 \cdot\cdot g_2, h_1 \cdot\cdot h_2}$;
$R_2 \leftarrow S'_{g_2+1 \cdot\cdot g_3, h_1 \cdot\cdot h_2}$;
$AT_{S,I}(S'_{g_1 \cdot\cdot g_3, h_1 \cdot\cdot h_2}, *) \leftarrow$ undefined;
$Q \leftarrow$ all disks in $\mathcal{D}$ of level $j$ intersecting the boundary of $R_1$ or $R_2$, i.e.,
    $\mathcal{D}^{\partial R_1}_{=j} \cup \mathcal{D}^{\partial R_2}_{=j}$;
for all $U \subseteq Q$ such that $|U| \leq 2C'k^2$ do
    if the disks in $I \cup U$ are disjoint then
        for $i = 1$ to 2 do
            $U_i \leftarrow \{D \in U \mid D$ intersects the boundary of $R_i\}$;
            $X_i \leftarrow AT_{S,I}(R_i, U_i)$;
        od;
        $X \leftarrow X_1 \cup X_2 \cup \{D \in Q \setminus U \mid D$ does not intersect the boundary of
            $S'_{g_1 \cdot\cdot g_3, h_1 \cdot\cdot h_2}\}$;
        $J \leftarrow \{D \in U \mid D$ intersects the boundary of $S'_{g_1 \cdot\cdot g_3, h_1 \cdot\cdot h_2}\}$;
        if $AT_{S,I}(S'_{g_1 \cdot\cdot g_3, h_1 \cdot\cdot h_2}, J)$ is undefined or
          $w(X) < w(AT_{S,I}(S'_{g_1 \cdot\cdot g_3, h_1 \cdot\cdot h_2}, J)$ then
            $AT_{S,I}(S'_{g_1 \cdot\cdot g_3, h_1 \cdot\cdot h_2}, J) \leftarrow X$;
        fi
    fi
od
```

FIG. 3.2. *Computing the auxiliary table $AT_{S,I}(S'_{g_1 \cdot\cdot g_3, h_1 \cdot\cdot h_2}, *)$ for MWVC.*

$\mathcal{D}^S$ is the set of all disks in $\mathcal{D}$ that intersect $S$ and that $\mathcal{D}^S_{<j}$ denotes the set of disks in $\mathcal{D}^S$ whose level is smaller than $j$, and similarly for $\mathcal{D}^S_{\leq j}$, $\mathcal{D}^S_{=j}$, $\mathcal{D}^S_{\geq j}$, and $\mathcal{D}^S_{>j}$. Let $I$ be a set of disjoint disks in $\mathcal{D}^S_{<j}$.

Assume that the entries of all tables $T_{S'}$ computed for previously processed squares $S'$ satisfy Property 1. Then each set $T_{S'_{g,h}}(I' \cup U)$ that is looked up by the algorithm of Figure 3.1 is such that

$$(\mathcal{D}^{S'_{g,h}}_{\leq j} \setminus (I' \cup U)) \cup T_{S'_{g,h}}(I' \cup U)$$

is a pseudocover of $S'_{g,h}$. Therefore, the set stored in $AT_{S,I}(S'_{g,h}, J)$ for $J = \{D \in U \mid D$ intersects the boundary of $S'_{g,h}\}$ satisfies Property 2.

Now assume that the information from table entries $AT_{S,I}(R_1, *)$ and $AT_{S,I}(R_2, *)$ is combined to obtain the table entries $AT_{S,I}(R_1 \cup R_2, *)$ using the algorithm of Figure 3.2. Consider some iteration of the algorithm of Figure 3.2 and let $U_1$ and $U_2$ be defined as calculated by the algorithm. Note that $U_1 \cup U_2 = U$ but $U_1 \cap U_2$ can be nonempty. Since table entries $AT_{S,I}(R_i, U_i)$ for $i = 1, 2$ satisfy Property 2, we have that the set

$$(\mathcal{D}^S_{<j} \setminus I) \cup ((\mathcal{D}^{\partial R_1}_{=j} \cup \mathcal{D}^{\partial R_2}_{=j}) \setminus (U_1 \cup U_2)) \cup AT_{S,I}(R_1, U_1) \cup AT_{S,I}(R_2, U_2)$$

is indeed a pseudocover of $R_1$ and $R_2$ and thus also of $R_1 \cup R_2$. Therefore, the set stored in $AT_{S,I}(R_1 \cup R_2, J)$ for $J = \{D \in U \mid D \text{ intersects the boundary of } R_1 \cup R_2\}$ satisfies Property 2 as well.

When the auxiliary table entries $AT_{S,I}(S, *)$ have been computed, the algorithm obtains $T_S(I)$ by taking the minimum weight set obtainable as $AT_{S,I}(S, J) \cup (\mathcal{D}^{\partial S}_{=j} \setminus J)$. Since the table entry $AT_{S,I}(S, J)$ leading to the minimum satisfies Property 2, the resulting table entry $T_S(I)$ satisfies Property 1.

In the end, the algorithm outputs a union of pseudocovers $T_S(\emptyset)$ in all relevant squares $S$ without parent. As every intersection of two disks is contained in some relevant square without parent, the algorithm always outputs a vertex cover of $\mathcal{D}$. □

**3.2. Analysis of approximation ratio.** Let $\mathcal{C}$ be any optimal vertex cover of $\mathcal{D}$. Every disk hits at most one vertical line on its level and at most one horizontal line on its level. For any pair of values $r$ and $s$, $0 \le r, s < k$, let $\mathcal{C}(r, s)$ be the set of all disks in $\mathcal{C}$ that hit a vertical line on their level whose index modulo $k$ equals $r$ or a horizontal line on their level whose index modulo $k$ equals $s$. Note that there must be a value of $r$ such that the total weight of disks in $\mathcal{C}$ that hit a vertical line on their level whose index modulo $k$ equals $r$ is at most $\frac{1}{k}w(\mathcal{C})$. An analogous argument shows that there is a value of $s$ with the corresponding property for horizontal lines. Therefore, there exist values of $r$ and $s$ such that the total weight of $\mathcal{C}(r, s)$ is at most $\frac{2}{k}$ times the weight of $\mathcal{C}$. Consider the subdivision of the plane that results from this choice of $r$ and $s$.

Let $\mathcal{R}$ be the set of all relevant squares. For any $j$-square $S$, let $\mathcal{C}(S)$ denote the disks in $\mathcal{C}$ that intersect $S$ and that are on level $j$. Note that $\mathcal{C} = \bigcup_{S \in \mathcal{R}} \mathcal{C}(S)$ but the sets $\mathcal{C}(S)$ and $\mathcal{C}(S')$ for $S \ne S'$ need not be disjoint.

LEMMA 3.2. *We have* $\sum_{S \in \mathcal{R}} w(\mathcal{C}(S)) \le (1 + \frac{6}{k})w(\mathcal{C})$.

*Proof.* The only disks in $\mathcal{C}$ that are counted more than once in the sum on the left-hand side are those that intersect several squares on their level. However, any disk can intersect at most 4 squares on its level. Furthermore, only disks in $\mathcal{C}(r, s)$ can intersect more than 1 square on their level. Thus, only disks of total weight $w(\mathcal{C}(r, s)) \le \frac{2}{k}w(\mathcal{C})$ are counted multiple times, and each disk is counted at most four times on the left-hand side, while it is counted once in $w(\mathcal{C})$. Thus $\sum_{S \in \mathcal{R}} w(\mathcal{C}(S))$ can exceed $w(\mathcal{C})$ by at most $3 \cdot \frac{2}{k}w(\mathcal{C})$, establishing the claimed inequality. □

Lemma 3.2 shows that the sum of the weights of the sets $\mathcal{C}(S)$ is only slightly larger than the weight of $\mathcal{C}$, although certain disks are counted several times in this sum. The following lemma shows that the weight of the vertex cover output by our algorithm does not exceed this sum. Intuitively, the lemma means that, on average, for each square $S$ on some level $j$ the weight of the set of disks from $\mathcal{D}^S_{=j}$ chosen by the algorithm does not exceed the weight of the disks chosen from $\mathcal{D}^S_{=j}$ in the optimal solution.

LEMMA 3.3. *Let $A$ be the vertex cover output by the algorithm. Then* $w(A) \le \sum_{S \in \mathcal{R}} w(\mathcal{C}(S))$.

*Proof.* For any relevant $j$-square $S$ or any rectangle $R$ of child squares of $S$, let $\mathcal{C}^S$ and $\mathcal{C}^R$ be the set of all disks in $\mathcal{C}$ that intersect $S$ and $R$, respectively. Define $\mathcal{C}^S_{<j}$, $\mathcal{C}^S_{\le j}$, $\mathcal{C}^S_{=j}$, etc. as usual. Observe that $\mathcal{C}(S) = \mathcal{C}^S_{=j}$. We claim that, after $T_S$ has been computed, we have

$$w(T_S(\mathcal{D}^S_{<j} \setminus \mathcal{C}^S_{<j})) \le \sum_{S' : S' \prec S} w(\mathcal{C}(S')) , \tag{3.1}$$

where $S' \prec S$ means that $S'$ is a relevant square that is contained in $S$. Note that $S \prec S$.

The proof is by induction on the number of relevant squares that have been processed by the algorithm. Assume that the algorithm is about to process the relevant square $S$ and that (3.1) holds for all squares that have been processed before $S$. One of the independent sets $I$ in $\mathcal{D}^S_{<j}$ enumerated by the algorithm is equal to $\mathcal{D}^S_{<j} \setminus \mathcal{C}^S_{<j}$. Consider that set $I$ in the following.

We turn to the computation of the auxiliary table entries $AT_{S,I}(R, J)$, where $R = S'_{g_1\cdots g_2,h_1\cdots h_2}$ is a rectangle of subsquares of $S$. We claim that when the table entries $AT_{S,I}(R, *)$ are computed, we have

$$w(AT_{S,I}(R,J)) \ \le \ w(\{D \in \mathcal{C}^R_{=j} \mid D \text{ is contained in } R\}) + \textstyle\sum_{S':S'\prec R, S'\neq S} w(\mathcal{C}(S'))$$
$$\text{for } J = \{D \in \mathcal{D}^R_{=j} \setminus \mathcal{C} \mid D \text{ intersects the boundary of } R\} = \mathcal{D}^{\partial R}_{=j} \setminus \mathcal{C}. \tag{3.2}$$

Note that the sum in (3.2) is over all relevant $j'$-squares, $j' > j$, that are contained in $R$, and that the sum does *not* include the term $w(\mathcal{C}(S))$ even if $R = S$.

First, consider the computation of $AT_{S,I}(S'_{g,h}, *)$ by the algorithm of Figure 3.1. In one of the iterations of the for-loop, the set $U$ is equal to $Q \setminus \mathcal{C} = \mathcal{D}^{S'_{g,h}}_{=j} \setminus \mathcal{C}$. For this set $U$, the table entry $T_{S'_{g,h}}(I' \cup U) = T_{S'_{g,h}}(\mathcal{D}^{S'_{g,h}}_{\le j} \setminus \mathcal{C}^{S'_{g,h}}_{\le j})$ is looked up. Since (3.1) holds for $T_{S'_{g,h}}$, we get that $w(T_{S'_{g,h}}(I' \cup U)) \le \sum_{S':S'\prec S'_{g,h}} w(\mathcal{C}(S'))$. Hence, the set

$$X = T_{S'_{g,h}}(I' \cup U) \cup \{D \in \mathcal{D}^{S'_{g,h}}_{=j} \setminus U \mid D \text{ is contained in } S'_{g,h}\}$$

stored in $AT_{S,I}(S'_{g,h}, J)$ for

$$\begin{aligned} J &= \{D \in U \mid D \text{ intersects the boundary of } S'_{g,h}\} \\ &= \{D \in \mathcal{D}^{S'_{g,h}}_{=j} \setminus \mathcal{C} \mid D \text{ intersects the boundary of } S'_{g,h}\} \end{aligned}$$

has weight at most

$$\begin{aligned} &\sum_{S':S'\prec S'_{g,h}} w(\mathcal{C}(S')) + w(\{D \in \mathcal{D}^{S'_{g,h}}_{=j} \setminus U \mid D \text{ is contained in } S'_{g,h}\}) \\ = &\sum_{S':S'\prec S'_{g,h}} w(\mathcal{C}(S')) + w(\{D \in \mathcal{C}^{S'_{g,h}}_{=j} \mid D \text{ is contained in } S'_{g,h}\}). \end{aligned}$$

From this we see that (3.2) is satisfied for $R = S'_{g,h}$.

Next, consider the combination of table entries $AT_{S,I}(R_1, *)$ and $AT_{S,I}(R_2, *)$ to obtain table entries $AT_{S,I}(R_1 \cup R_2, *)$ using the algorithm of Figure 3.2. In one of the iterations of the for-loop, the set $U$ is equal to the set

$$Q\setminus\mathcal{C} = \{D \in \mathcal{D}^{R_1\cup R_2}_{=j}\setminus\mathcal{C} \mid D \text{ intersects the boundary of } R_1 \text{ or } R_2\} = (\mathcal{D}^{\partial R_1}_{=j} \cup \mathcal{D}^{\partial R_2}_{=j})\setminus\mathcal{C}.$$

For this set $U$, the table entries $AT_{S,I}(R_i, U_i)$ are looked up for $i = 1, 2$, where $U_i = \{D \in U \mid D \text{ intersects the boundary of } R_i\}$. Thus, we have $U_i = \mathcal{D}^{\partial R_i}_{=j} \setminus \mathcal{C}$ and we know that (3.2) holds for table entries $AT_{S,I}(R_i, U_i)$. Then the weight of the set

$$\begin{aligned}X &= AT_{S,I}(R_1,U_1)\cup AT_{S,I}(R_2,U_2)\cup\{D\in(\mathcal{D}^{\partial R_1}_{=j}\cup\mathcal{D}^{\partial R_2}_{=j})\setminus U \mid D \text{ is contained}\\ &\qquad \text{in } R_1\cup R_2\}\\ &= AT_{S,I}(R_1,U_1)\cup AT_{S,I}(R_2,U_2)\cup\{D\in\mathcal{C}\cap(\mathcal{D}^{\partial R_1}_{=j}\cup\mathcal{D}^{\partial R_2}_{=j}) \mid D \text{ is contained}\\ &\qquad \text{in } R_1\cup R_2\}\\ &= AT_{S,I}(R_1,U_1)\cup AT_{S,I}(R_2,U_2)\cup\{D\in\mathcal{C}^{\partial R_1}_{=j}\cup\mathcal{C}^{\partial R_2}_{=j} \mid D \text{ is contained in } R_1\cup R_2\}\\ &= AT_{S,I}(R_1,U_1)\cup AT_{S,I}(R_2,U_2)\cup\{D\in\mathcal{C}^{\partial R_1}_{=j}\cap\mathcal{C}^{\partial R_2}_{=j} \mid D \text{ is contained in } R_1\cup R_2\}\\ &= AT_{S,I}(R_1,U_1)\cup AT_{S,I}(R_2,U_2)\cup\{D\in\mathcal{C}^{\partial R_1\cap\partial R_2}_{=j} \mid D \text{ is contained in } R_1\cup R_2\}\end{aligned}$$

is at most

$$\begin{aligned}&w(\{D\in\mathcal{C}^{R_1}_{=j}\mid D \text{ is contained in } R_1\})+\sum_{S':S'\prec R_1}w(\mathcal{C}(S'))\\ &\quad+w(\{D\in\mathcal{C}^{R_2}_{=j}\mid D \text{ is contained in } R_2\})+\sum_{S':S'\prec R_2}w(\mathcal{C}(S'))\\ &\quad+w(\{D\in\mathcal{C}^{\partial R_1\cap\partial R_2}_{=j}\mid D \text{ is contained in } R_1\cup R_2\})\\ &=\left(\sum_{S':S'\prec(R_1\cup R_2),S'\neq S}w(\mathcal{C}(S'))\right)+w(\{D\in\mathcal{C}^{R_1\cup R_2}_{=j}\mid D \text{ is contained in } R_1\cup R_2\}).\end{aligned}$$

The set $X$ is stored in $AT_{S,I}(R_1\cup R_2,J)$ for $J=\{D\in U\mid D$ intersects the boundary of $R_1\cup R_2\}=\{D\in\mathcal{D}^{R_1\cup R_2}_{=j}\setminus\mathcal{C}\mid D$ intersects the boundary of $R_1\cup R_2\}$ provided its weight is smaller than that of the previously stored set. This shows that (3.2) holds for $R=R_1\cup R_2$ when the algorithm of Figure 3.2 terminates. Thus, by induction we see that all table entries $AT_{S,I}(R,J)$ satisfy (3.2).

This implies that for $J=\mathcal{D}^{\partial S}_{=j}\setminus\mathcal{C}$, the table entry $AT_{S,I}(S,J)$ has weight at most $w(\{D\in\mathcal{C}^S_{=j}\mid D$ is contained in $S\})+\sum_{S':S'\prec S,S'\neq S}w(\mathcal{C}(S'))$. Note that $\mathcal{D}^{\partial S}_{=j}\setminus J=\mathcal{C}^{\partial S}_{=j}$. The set $AT_{S,I}(S,J)\cup(\mathcal{D}^{\partial S}_{=j}\setminus J)$, which is one of the candidates for the table entry $T_S(I)$, has weight at most $w(\mathcal{C}(S))+\sum_{S':S'\prec S,S'\neq S}w(\mathcal{C}(S'))$. Thus, we get that (3.1) holds for $T_S$ as well, and the inductive step with respect to the tables $T_S$ is established.

Finally, let $\mathcal{R}_0$ be the set of all relevant squares without parent. The algorithm outputs $\bigcup_{S\in\mathcal{R}_0}T_S(\emptyset)$ as a solution. For any relevant $j$-square $S\in\mathcal{R}_0$, we have $\mathcal{C}^S_{<j}=\emptyset$. Thus we get from (3.1) that $\sum_{S\in\mathcal{R}_0}w(T_S(\emptyset))\leq\sum_{S\in\mathcal{R}}w(\mathcal{C}(S))$. ☐

Combining these lemmas, we get that the algorithm outputs a vertex cover whose weight is at most a factor of $1+\frac{6}{k}$ larger than the weight of the optimal vertex cover. Furthermore, by a similar analysis as in the case of MWIS, the running-time of the algorithm is bounded by $n^{O(k^2)}$. Thus we obtain our second main theorem.

THEOREM 3.4. *There is a PTAS for MWVC in disk graphs, provided that a disk representation of the graph is given. The running-time for achieving approximation ratio* $1+\varepsilon$ *is* $n^{O(1/\varepsilon^2)}$ *for a disk graph with* $n$ *disks.*

## 4. Extension to other geometric intersection graphs.

In the previous sections we have presented PTASs for MWIS and MWVC in the intersection graphs of disks with arbitrary diameters. Our approach does not make use of any specific properties of disks; it is required only that the given geometric objects can be partitioned into levels such that only a constant number (for fixed $k$) of disjoint objects of level smaller than $j$ can intersect a square on level $j$. Therefore, the same approach can be used for other geometric objects such as squares or regular polygons. We can also deal with rectangles if the ratio between the height and the width does not differ by more than a constant factor between different rectangles, because it suffices to scale the input along one axis so that the resulting rectangles are approximately squares.

Furthermore, the approach works for geometric objects in any $d$-dimensional space provided that $d$ is a constant. The space is partitioned into $d$-dimensional cubes on each level, and instead of removing objects (in the case of MWIS) that hit certain horizontal or vertical lines, we remove objects that hit certain hyperplanes.

More specifically, to obtain PTASs for MWIS and MWVC in the intersection graphs of geometric objects in $d$ dimensions, the following conditions are sufficient:

(i) $d$ is a constant.

(ii) For each object $i$, a $d$-dimensional ball $B_i$ that contains $i$ can be determined in polynomial time.

(iii) Let level $j$ contain all given objects whose balls $B_i$ have diameter $d_i$ satisfying $(k+1)^{-j} \geq d_i > (k+1)^{-(j+1)}$. Then the number of disjoint objects of level smaller than $j$ that can intersect a $d$-dimensional cube with side length $k(k+1)^{-j}$ must be bounded by a constant (for fixed $k$), and the number of disjoint objects of level $j$ that can intersect the boundary of a $d$-dimensional hyperrectangle with sides of length at most $k(k+1)^{-j}$ must be bounded by a constant (for fixed $k$).

(iv) One can decide in polynomial time whether two of the given objects intersect and whether an object intersects an arbitrary cube.

For example, our approach yields a PTAS for MWIS or MWVC in the intersection graphs of $n$ balls in $d$-dimensional space with running-time $n^{O(1/\varepsilon^{2d-2})}$ for any constant $d$.

## 5. Conclusion and open problems.

We have presented the first known PTASs for MWIS and MWVC in the intersection graphs of disks. Our algorithms partition the given disks into levels and apply a shifting strategy on all levels simultaneously. The PTASs can be generalized to other "disk-like" geometric objects in $d$ dimensions, achieving running-time $n^{O(1/\varepsilon^{2d-2})}$ for any constant $d$. Another PTAS for MWIS in disk graphs, based on ideas similar to ours but using shifted quadtrees in the recursive subdivision strategy, was discovered recently by Chan [8]. Chan's algorithm achieves running-time $n^{O(1/\varepsilon)}$ for MWIS in disk graphs and $n^{O(1/\varepsilon^{d-1})}$ for the $d$-dimensional generalization. MWVC is not considered in Chan's paper.

Our approximation schemes use a partitioning of the plane into squares on each level, and this works only if the ratio of the height to the width is roughly the same for all given geometric objects. It is not clear whether the approach can be extended to the intersection graphs of arbitrary axis-parallel rectangles, for example. For computing a maximum independent set among $n$ given rectangles, an $O(\log n)$-approximation algorithm was presented by Agarwal, van Kreveld, and Suri [1]. Berman et al. [4] improved upon this result and gave a family of approximation algorithms with approximation ratio $1 + \frac{\log n}{c}$ for *any* positive constant $c$. It is an open problem to devise an approximation algorithm with constant approximation ratio or even a PTAS for this problem, or to provide evidence that MWIS in intersection graphs of arbitrary rectangles is substantially harder to approximate than in intersection graphs of squares or disks.

Concerning disk graphs, it would be interesting to see whether one can also find a PTAS for the minimum dominating set problem, where the goal is to select a minimum number of disks such that each given disk is either selected or intersects a disk that is selected. For planar graphs and for unit disk graphs, PTASs for the minimum dominating set problem have been found using the shifting strategy [2, 18], but we have not yet been able to adapt the approach of the present paper to the minimum dominating set problem.

The PTASs for disk graphs require that the disk representation of the graph is given as part of the input. It would be interesting to determine whether a PTAS can be obtained even in the case without given representation. In this context we note that for the maximum clique problem in unit disk graphs, which can be solved in polynomial time [9], an exact algorithm that does not need the representation has recently been found by Raghavan and Spinrad [26]. This is somewhat surprising since it is $\mathcal{NP}$-hard to determine whether a given graph is a unit disk graph [7]. In fact, the algorithm by Raghavan and Spinrad is *robust* in the sense that for any given graph $G$, the algorithm either finds a maximum clique in $G$ or asserts correctly that $G$ is not a unit disk graph. Robust algorithms solving MWIS optimally in certain classes of graphs that are characterized by forbidden subgraphs have been presented by Brandstädt [5]. The concept of robustness can be applied to approximation algorithms for graph problems by calling an algorithm a *robust $\rho$-approximation algorithm* for a graph class $\mathcal{C}$ if, for any input graph $G$, the algorithm either outputs a solution that is within a factor $\rho$ of the optimal solution or asserts correctly that $G$ is not a member of $\mathcal{C}$. Recently, a robust PTAS that does not require the disk representation as part of the input has been obtained for MWIS in unit disk graphs by Nieberg, Hurink, and Kern [25].

## REFERENCES

[1] P. K. Agarwal, M. van Kreveld, and S. Suri, *Label placement by maximum independent set in rectangles*, Comput. Geom. 11 (1998), pp. 209–218.

[2] B. S. Baker, *Approximation algorithms for NP-complete problems on planar graphs*, J. ACM, 41 (1994), pp. 153–180.

[3] R. Bar-Yehuda and S. Even, *A local-ratio theorem for approximating the weighted vertex cover problem*, Ann. Discrete Math., 25 (1985), pp. 27–45.

[4] P. Berman, B. DasGupta, S. Muthukrishnan, and S. Ramaswami, *Improved approximation algorithms for rectangle tiling and packing*, in Proceedings of the 12th Annual ACM–SIAM Symposium on Discrete Algorithms (SODA'01), ACM, New York, SIAM, Philadelphia, 2001, pp. 427–436.

[5] A. Brandstädt, *On robust algorithms for the maximum weight stable set problem*, in Proceedings of the 13th International Symposium on Fundamentals of Computation Theory (FCT 2001), Lecture Notes in Comput. Sci. 2138, Springer-Verlag, Berlin, 2001, pp. 445–458.

[6] A. Brandstädt, V. B. Le, and J. P. Spinrad, *Graph Classes: A Survey*, SIAM Monogr. Discrete Math. Appl. 3, SIAM, Philadelphia, 1999.

[7] H. Breu and D. G. Kirkpatrick, *Unit disk graph recognition is NP-hard*, Comput. Geom. 9 (1998), pp. 3–24.

[8] T. M. Chan, *Polynomial-time approximation schemes for packing and piercing fat objects*, J. Algorithms, 46 (2003), pp. 178–189.

[9] B. N. Clark, C. J. Colbourn, and D. S. Johnson, *Unit disk graphs*, Discrete Math., 86 (1990), pp. 165–177.

[10] S. Doddi, M. V. Marathe, A. Mirzaian, B. M. E. Moret, and B. Zhu, *Map labeling and its generalizations*, in Proceedings of the 8th Annual ACM–SIAM Symposium on Discrete Algorithms (SODA'97), ACM, New York, SIAM, Philadelphia, 1997, pp. 148–157.

[11] S. Doddi, M. V. Marathe, and B. M. E. Moret, *Point set labeling with specified positions*, Internat. J. Comput. Geom. Appl., 12 (2002), pp. 29–66.

[12] M. C. Golumbic, *Algorithmic Graph Theory and Perfect Graphs*, Academic Press, New York, 1980.

[13] W. K. Hale, *Frequency assignment: Theory and applications*, Proceedings of the IEEE, 68 (1980), pp. 1497–1514.

[14] J. Håstad, *Clique is hard to approximate within $n^{1-\epsilon}$*, Acta Math., 182 (1999), pp. 105–142.

[15] J. Håstad, *Some optimal inapproximability results*, in Proceedings of the 29th Annual ACM Symposium on Theory of Computing (STOC'97), ACM Press, New York, 1997, pp. 1–10.

[16] P. Hliněný and J. Kratochvíl, *Representing graphs by disks and balls*, Discrete Math., 229 (2001), pp. 101–124.

[17] D. S. Hochbaum and W. Maass, *Approximation schemes for covering and packing problems in image processing and VLSI*, J. ACM, 32 (1985), pp. 130–136.

[18] H. B. Hunt III, M. V. Marathe, V. Radhakrishnan, S. S. Ravi, D. J. Rosenkrantz, and R. E. Stearns, *NC-approximation schemes for NP- and PSPACE-hard problems for geometric graphs*, J. Algorithms, 26 (1998), pp. 238–274.

[19] K. Jansen, *Approximate strong separation with application in fractional graph coloring and preemptive scheduling*, Theoret. Comput. Sci., 302 (2003), pp. 239–256.

[20] K. Jansen and L. Porkolab, *On preemptive resource constrained scheduling: Polynomial-time approximation schemes*, in Proceedings of the 9th Conference on Integer Programming and Combinatorial Optimization (IPCO), Lecture Notes in Comput. Sci. 2337, Springer-Verlag, Berlin, 2002, pp. 329–349.

[21] P. Koebe, *Kontaktprobleme der konformen Abbildung*, Berichte über die Verhandlungen der Sächsischen Akademie der Wissenschaften, Leipzig, Math.-Phys. Klasse, 88 (1936), pp. 141–164.

[22] E. Malesińska, *Graph-Theoretical Models for Frequency Assignment Problems*, Ph.D. thesis, Technische Universität Berlin, Berlin, Germany, 1997.

[23] M. V. Marathe, H. Breu, H. B. Hunt III, S. S. Ravi, and D. J. Rosenkrantz, *Simple heuristics for unit disk graphs*, Networks, 25 (1995), pp. 59–68.

[24] T. A. McKee and F. R. McMorris, *Topics in Intersection Graph Theory*, SIAM Monogr. Discrete Math. Appl. 2, SIAM, Philadelphia, 1999.

[25] T. Nieberg, J. L. Hurink, and W. Kern, *A robust PTAS for maximum independent sets in unit disk graphs*, in Proceedings of the 30th International Workshop on Graph-Theoretic Concepts in Computer Science (WG'04), Lecture Notes in Comput. Sci. 3353, Springer-Verlag, New York, 2004, pp. 214–221.

[26] V. Raghavan and J. Spinrad, *Robust algorithms for restricted domains*, in Proceedings of the 12th Annual ACM–SIAM Symposium on Discrete Algorithms (SODA'01), ACM, New York, SIAM, Philadelphia, 2001, pp. 460–467.

[27] M. van Kreveld, T. Strijk, and A. Wolff, *Point labeling with sliding labels*, Comput. Geom., 13 (1999), pp. 21–47.

[28] A. Wolff and T. Strijk, *The map-labeling bibliography*, http://i11www.ilkd.uni-karlsruhe.de/∼awolff/map-labeling/bibliography/ (2003).

SIAM J. COMPUT.
Vol. 34, No. 6, pp. 1324–1330

# QUANTUM ALGORITHMS FOR ELEMENT DISTINCTNESS*

HARRY BUHRMAN†, CHRISTOPH DÜRR‡, MARK HEILIGMAN§, PETER HØYER¶, FRÉDÉRIC MAGNIEZ‖, MIKLOS SANTHA‖, AND RONALD DE WOLF†

**Abstract.** We present several applications of quantum amplitude amplification for deciding whether all elements in the image of a given function are distinct, for finding an intersection of two sorted tables, and for finding a triangle in a graph. Our techniques generalize and improve those of Brassard, Høyer, and Tapp [*ACM SIGACT News*, 28 (1997), pp. 14–19]. This shows that in the quantum world element distinctness is significantly easier than sorting, in contrast to the classical world.

**Key words.** quantum query complexity, element distinctness, collision problem

**AMS subject classification.** 68Q25

**DOI.** 10.1137/S0097539702402780

**1. Introduction.** In the last decade, quantum computing has become a prominent and promising area of theoretical computer science. Realizing this promise will require two things: actually building a quantum computer and discovering tasks where a quantum computer is significantly faster than a classical computer. Here we are concerned with the second issue. Few good quantum algorithms are known to date. The two main examples are Shor's algorithm for factoring [23], which achieves an exponential speed-up over the best known classical factoring algorithms, and Grover's search algorithm [15], which achieves a quadratic speed-up over classical search algorithms. Whereas the first so far has remained a seminal but somewhat isolated result, the second has been applied as a building block in quite a few other quantum algorithms [6, 8, 9, 10, 21, 20, 7, 12].

The security of the widely used cryptosystem RSA is based on the assumption that it is hard to factor integers. Shor's algorithm solves precisely this task. In the same vein, the security of digital signatures is based on the assumption that it is difficult to find two items that map to the same value for some particular function. This motivates the research on the quantum complexity of this task. We define different variants of this problem. Though we do not improve the bounds for the following problem, we define it first to start our explanation. We use $[N]$ to denote $\{1, \ldots, N\}$.

*Received by the editors February 19, 2002; accepted for publication (in revised form) December 10, 2004; published electronically June 27, 2005. A preliminary version of this paper appeared in *Proceedings of the* 16th *IEEE Conference on Computational Complexity*, 2001, pp. 131–137. This research was partially supported by the EU 5th framework programs QAIP IST-1999-11234 and RAND-APX IST-1999-14036.

http://www.siam.org/journals/sicomp/34-6/40278.html

†CWI, P.O. Box 94079, Amsterdam, The Netherlands (buhrman@cwi.nl, rdewolf@cwi.nl). The first author is also affiliated with the University of Amsterdam.

‡Université Paris-Sud, LRI, 91405 Orsay, France (durr@lri.fr).

§NSA, Suite 6111, Fort George G. Meade, MD 20755 (miheili@nsa.gov).

¶Department of Computer Science, University of Calgary, Calgary T2N 1N4, AB, Canada (hoyer@cpsc.ucalgary.ca). This research was conducted while the author was at BRICS, University of Aarhus, Denmark.

‖CNRS–LRI, UMR 8623 Université Paris-Sud, 91405 Orsay, France (magniez@lri.fr, santha@lri.fr).

Collision Problem

**input** $f : [N] \to [M]$ which is 2-to-1, i.e., $\forall i \in [N] \exists! j \in [N],\ i \neq j : f(i) = f(j)$

**output** $i, j \in [N]$ with $i \neq j$ and $f(i) = f(j)$

**complexity** Classically, the bounded-error query complexity is $\Theta(N^{1/2})$. For a quantum computer the bounded-error query complexity is $\Theta(N^{1/3})$: In 1997 Brassard, Høyer, and Tapp [8] gave a bounded-error quantum algorithm using $O(N^{1/3})$ queries to $f$, and in 2002 Aaronson and Shi [1] showed the matching lower bound.

In the following problem we remove the assumption about the input. The birthday paradox gives a simple relation between both problems. A random subset of size $\sqrt{N}$ of the domain of any 2-to-1 function contains with high probability a collision pair. Therefore any bounded-error algorithm for Element Distinctness using $O(N^{\alpha})$ queries implies a bounded-error algorithm for the Collision Problem using $O(N^{\alpha/2})$ queries.

Element Distinctness

**input** $f : [N] \to [M]$

**output** $i, j \in [N]$ with $i \neq j$ and $f(i) = f(j)$, or "all distinct" if $f$ is injective

**complexity** We present a bounded-error quantum algorithm which makes $O(N^{3/4})$ queries. It dates from early 2000 and first appeared in [11]. However, recently the bounded-error quantum query complexity was shown to be $\Theta(N^{2/3})$: The lower bound follows from Aaronson and Shi [1] by the observation above, and an algorithm matching this bound was found in 2003 by Ambainis [3] using a quantum walk. The classical bounded-error query complexity is $\Theta(N)$ by a trivial reduction from the OR-problem: For an OR-instance $x \in \{0,1\}^N$ we define the function $f : [N+1] \to [N+1]$, where $f(N+1) = 0$ and for all $i \in [N]$ $f(i) = (1 - x_i)i$. Now $\mathrm{OR}(x) = 1$ if and only if $f$ contains a collision pair.

The element distinctness problem has been well studied classically [24, 18, 14, 5]. It is particularly interesting because its classical complexity is related to that of sorting, which is well known to require $N \log N + \Theta(N)$ comparisons in the classical world. If we sort $f$, we can decide element distinctness by going through the sorted list once, which gives a classical upper bound of $N \log N + O(N)$ comparisons. Conversely, element distinctness requires $\Omega(N \log N)$ comparisons in the case of classical bounded-error algorithms (even in a much stronger model [14]), so sorting and element distinctness are essentially equally hard classically. On a quantum computer, the best known upper bound for sorting is $0.53\, N \log N$ comparisons [13], and such a linear speed-up is best possible: quantum sorting requires $\Omega(N \log N)$ comparisons, even if one allows a small probability of error [16]. Accordingly, our $O(N^{3/4} \log N)$ upper bound shows that element distinctness is significantly easier than sorting for a quantum computer, in contrast to the classical case.

In this paper we also give algorithms for related problems. Typically, web search engines like Google associate to every word a list of pages containing it (sorted in order of its page rank) and when the query is "Rolling Stones," for example, then the search engine must output the intersection of the lists associated to the words "Rolling" and "Stones." Now imagine a search engine implemented on a quantum computer. This motivates the following problem.

List Intersection

**input** $f, g : [N] \to [M]$ each is monotone increasing

**output** $i, j \in [N]$ with $i \neq j$ and $f(i) = f(j)$, or "lists disjoint" if the images of $f$ and $g$ are disjoint

**complexity** We present a bounded-error quantum algorithm which makes $O(\sqrt{N} c^{\log^* N})$ queries to $f$ for some constant $c > 1$. A trivial lower bound $\Omega(\sqrt{N})$ can be obtained by a reduction from the OR-problem: given an OR-instance $x \in \{0,1\}^N$, define $f, g : [N] \to [2N+1]$ by $f(i) = 2i + 1$ and $g(i) = 2i + x_i$ for all $i \in [N]$. Then $f$ and $g$ are ordered, and $\mathrm{OR}(x) = 1$ iff the List Intersection problem has a solution. The same reduction shows that the classical bounded-error query complexity is $\Theta(N)$.

The function $\log^\star(N)$ is defined as the minimum number of iterated applications of the logarithm function necessary to obtain a number less than or equal to 1: $\log^\star(N) = \min\{i \geq 0 \mid \log^{(i)}(N) \leq 1\}$, where $\log^{(i)} = \log \circ \log^{(i-1)}$ denotes the $i$th iterated application of log, and $\log^{(0)}$ is the identity function. Even though $c^{\log^\star(N)}$ is exponential in $\log^\star(N)$, it is still very small in $N$, in particular $c^{\log^\star(N)} \in o(\log^{(i)}(N))$ for any constant $i \geq 1$.

To a function $f : [N] \to [M]$ we can associate a *collision graph* $G(V, E)$ with $V = [N]$ and $(i, j) \in E$ if $i \neq j$ and $f(i) = f(j)$. The Element Distinctness problem simply consists of finding an edge in $G$. An interesting problem is to ask whether $G$ contains some fixed subgraph. A simple, yet nontrivial subgraph is the triangle, i.e., the complete graph on three vertices.

Triangle Finding

**input** the symmetric adjacency matrix $M : [n] \times [n] \to \{0,1\}$ of a graph with $m$ edges

**output** $u, v, w \in [N]$ such that $M(u,v) = M(v,w) = M(w,u) = 1$, or "failure" if the graph contains no triangle

**complexity** We present a bounded-error quantum algorithm which needs $O(n + \sqrt{nm})$ queries. A better algorithm has been found in 2003, with $O(n^{1.3})$ bounded-error query quantum complexity [19], while Yao [25] showed a lower bound of $\Omega(n^{2/3} \log^{1/6} n)$. Classically a simple reduction from the OR-problem shows that the bounded-error query complexity is $\Theta(n^2)$, even if $m = O(n)$.

## 2. Preliminaries.

We assume the reader is familiar with the formalism of quantum computing; otherwise we refer to [22]. The quantum ingredient of our algorithms is *amplitude amplification* [7], which generalizes quantum search [15]. The essence of amplitude amplification can be summarized by the following theorem.

Theorem 2.1 (amplitude amplification). *There exists a quantum algorithm QSearch with the following property. Let $\mathcal{A}$ be any quantum algorithm that uses no measurements and that maps $|0\rangle$ to a superposition $\sum_{x \in X} \alpha_x |x\rangle$ for some set $X$. Let $g : X \to \{0,1\}$ be a function testing whether a basis state represents a* solution *or not. Let $p$ be the success probability of $\mathcal{A}$, i.e., $p^2 = \sum_{x : g(x)=1} |\alpha_x|^2$. Let $S_g$ be an operator implementing $g$ such that $S_g|x\rangle = (-1)^{g(x)}|x\rangle$ for every $x \in X$. Then algorithm QSearch finds a solution using an expected number of $O(1/\sqrt{p})$ applications of $\mathcal{A}$, $\mathcal{A}^{-1}$, and $S_g$ if $p > 0$, and otherwise runs forever.*

Note that when an algorithm $\mathcal{A}$ does make measurements during its computation there is a standard trick that transforms it into an equivalent algorithm $\mathcal{A}'$ that does not. We replace every measurement with an operator writing the value, which would be the result of the measurement, in a new register, which initially is all zero. In the rest of the computation, every computation depending on the result of the measurement will depend rather on the content of this register.

QSearch works by iterating the unitary transformation $Q = -\mathcal{A}S_0\mathcal{A}^{-1}S_g$ a number of times, starting with initial state $\mathcal{A}|0\rangle$. The operator $S_0$ is defined as $S_0|0\rangle = -|0\rangle$ and $S_0|x\rangle = |x\rangle$ for all $x \neq 0$. The analysis of [7] shows that a measurement after $\Theta(1/\sqrt{p})$ iterations of $Q$ yields a solution with probability close to 1. The algorithm QSearch does not need to know the value of $p$ in advance, but if $p$ is known, then a slight modification finds a solution *with certainty* using $O(1/\sqrt{p})$ applications of $\mathcal{A}$, $\mathcal{A}^{-1}$, and $S_g$.

Grover's algorithm for searching a space of $N$ items is a special case of amplitude amplification, where $\mathcal{A}$ is the Hadamard transform on each qubit. This $\mathcal{A}$ has probability $p \geq 1/N$ of finding a solution (if there is at least one), so amplitude amplification implies an $O(\sqrt{N})$ quantum algorithm for searching the space. We refer to this process as "quantum searching."

## 3. Element distinctness.

```
ALGORITHM: FIND A COLLISION PAIR IN $f : [N] \to [M]$
  1. Partition the domain of $f$ into disjoint sets $S_1, \dots, S_{\sqrt{N}}$ of size $O(\sqrt{N})$ each.
  2. Apply amplitude amplification to the following inner block:
     (a) Select a random subset $S_k$ of the partition.
     (b) Query all values $f(i)$ for $i \in S_k$, and build a binary search tree over the
         set $f(S_k) := \{f(i) : i \in S_k\}$. If $S_k$ contains a collision pair, output it.
     (c) Otherwise search $j \in [N]\backslash S_k$ such that $f(j) \in f(S_k)$. Use the quantum
         search procedure which succeeds with probability at least 1/2 provided
         $S_k$ contains one element of a collision pair. In case of success, output
         the collision pair.
```

THEOREM 3.1. *If $f$ has a collision pair $i, j$, then the previous algorithm finds it after an expected number of $O(N^{3/4})$ queries to $f$.*

*Proof.* With probability at least $1/\sqrt{N}$, step 2(a) selects a subset containing $i$ or $j$. Suppose this is the case. Then either the set contains a collision pair or it does not. If it does, then step 2(b) finds it, and if it does not, then with probability at least 1/2, step 2(c) finds a collision pair. Therefore amplitude amplification will iterate the inner loop $O(N^{1/4})$ expected times until it succeeds. Steps 2(b) and 2(c) each use $O(\sqrt{N})$ queries, from which we conclude the claimed complexity. ∎

A weaker model is the comparison model, where we are allowed to ask query $f(i) \leq f(j)$ only for given indices $i, j$, rather than for the actual values $f(i), f(j)$. The previous algorithm can be adapted to that model with the price of an $O(\log N)$ factor in steps 2(b) and 2(c). In contrast, for classical (exact or bounded-error) algorithms, element distinctness is as hard as sorting and requires $\Theta(N \log N)$ comparisons.

## 4. List intersection.

We are given two monotone increasing functions $f, g : [N] \to [M]$ and search for $i, j \in [N]$ such that $f(i) = g(j)$. A simple algorithm would be to make a quantum search for $i \in [N]$ such that there exists $j \in [N]$ with $f(i) = g(j)$. The quantum search of $i$ will need $O(\sqrt{N})$ iterations, and the binary search of $j$ will need $O(\log N)$ queries. This gives a bounded-error quantum algorithm using $O(\sqrt{N} \log N)$ queries. We now show how to get rid of most of the log factor by exploiting the fact that both functions are monotone increasing.

Our quantum algorithm solves the problem using $O\big(\sqrt{N} c^{\log^\star(N)}\big)$ comparisons for some constant $c > 0$. We define a set of subproblems such that the original problem

$(f, g)$ contains a collision pair if and only if at least one of the subproblems contains one. We then solve the original problem by running the subproblems in quantum parallel and applying amplitude amplification.

Let $1 \leq r < N$ be an integer. For the purpose of defining subproblems we extend the functions $f$ and $g$ to the domain $[1, N+r]$, mapping $f(N+i) = \max\{f(N), g(N)\}+i$ and $g(N+i) = f(N+i)+r$ for all $1 \leq i \leq r$ and extending at the same time the range of $f$ and $g$ to $[M+2r]$. We also define the *insertion point* of some integer $x < h(N+1)$ in a monotone increasing function $h : [N+r] \rightarrow [M+2r]$ as the smallest index $i$ such that $h(i) \geq x$.

We define $2\left\lceil \frac{N}{r} \right\rceil$ subproblems as follows. For each $0 \leq i \leq \lceil N/r \rceil - 1$, consider the subproblem $(f_i, g_i')$, where $f_i$ denotes the restriction of $f$ to subdomain $[ir+1, (i+1)r]$ and $g_i'$ denotes the restriction of $g$ to $[j, j+r-1]$, where $j$ is the insertion point of $f(ir+1)$ in $g$.

Similarly, for each $0 \leq j \leq \lceil N/r \rceil - 1$, consider the subproblem $(f_j', g_j)$, where $g_j$ denotes the restriction of $g$ to $[jr+1, (j+1)r]$ and $f_j'$ denotes the restriction of $f$ to $[i, i+r-1]$, where $i$ is the insertion point of $g(jr+1)$ in $f$.

Lemma 4.1. *If $i, j \in [N]$ is a collision pair for $(f, g)$, then it is also a collision pair for one of the subproblems.*

*Proof.* Let $k = \lfloor i/r \rfloor + 1$ and let $k'$ be the insertion point of $f(k)$ in $g$. If $j \in [k', k'+r-1]$, then $(i, j)$ is also a collision pair for the subproblem $(f_k, g_k')$. However, if $j \notin [k', k'+r-1]$, then let $\ell = \lfloor j/r \rfloor + 1$. We have $f(k) \leq g(\ell) \leq f(i)$. Therefore the insertion point $\ell'$ of $g(\ell)$ in $f$ satisfies $i \in [\ell', \ell'+r-1]$, from which we conclude that $(i, j)$ is a collision pair for the subproblem $(f_{\ell}', g_{\ell})$. $\square$

Theorem 4.2. *There exists a quantum algorithm that outputs a collision pair between $f$ and $g$ with probability at least $\frac{2}{3}$ provided one exists, using $O\left(\sqrt{N} c^{\log^\star(N)}\right)$ queries, for some constant $c > 1$.*

*Proof.* Let $T(N)$ denote the worst-case number of queries required if $f$ and $g$ have domain of size $N$. We show that

$$T(N) \leq c' \sqrt{\frac{N}{r}} \left(\lceil \log(N+1) \rceil + T(r)\right) \tag{4.1}$$

for some (small) constant $c'$. Let $0 \leq i \leq \lceil N/r \rceil - 1$ and consider the subproblem $(f_i, g_i')$. To find the insertion point of $f(\lfloor i/r \rfloor + 1)$ in $g$ we need $\lceil \log(N+1) \rceil$ queries by using binary search. Then we need at most $T(r)$ additional queries to find a collision pair for $(f_i, g_i')$. There are $2\left\lceil \frac{N}{r} \right\rceil$ subproblems, so by applying amplitude amplification we can find a collision pair among any one of them with probability at least $\frac{2}{3}$, provided there is one, using the number of queries claimed in (4.1).

We pick $r = \lceil \log^2(N) \rceil$. Since $T(r) \geq \Omega(\sqrt{r}) = \Omega(\log N)$, (4.1) implies

$$T(N) \leq c'' \sqrt{\frac{N}{r}} T(r) \tag{4.2}$$

for some constant $c''$. Furthermore, our choice of $r$ implies that the depth of the recursion defined by (4.2) is on the order of $\log^\star(N)$, so unfolding the recursion gives the theorem. $\square$

**5. Triangle finding.** Finally we consider a related search problem. Consider an undirected graph $G = (V, E)$ on $|V| = n$ nodes with $|E| = m$ edges. There are $N = \binom{n}{2}$ edge slots in $E$, which we can query in a black box fashion (see also [10, section 7]). The goal is now to find distinct vertices $a, b, c \in V$ such that $(a, b)$, $(a, c)$,

$(b,c) \in E$. Since there are $\binom{n}{3}$ triples $a,b,c$, and we can decide whether a given triple is a triangle using three queries, we can use Grover's algorithm to find a triangle in $O(n^{3/2})$ queries. Below we give an algorithm that has the same complexity for dense graphs $m = O(n^2)$ but is more efficient for sparse graphs. In particular when $m = O(n)$, then the algorithm uses only $O(n)$ queries, while any classical bounded-error algorithm needs $\Omega(n^2)$ queries by a sensitivity argument for distinguishing the star graph, with the same graph augmented by a single edge.

```
ALGORITHM: FIND A TRIANGLE
  1. Use the bounded-error quantum counting procedure from [7, Theorem 18] to
     get a factor-2 estimation $m'$ of the number of edges $m$, with $O(n)$ expected
     queries.
  2. Apply amplitude amplification to the following inner block, interrupting it
     after $O(\sqrt{m'})$ calls to the inner block.
     (a) Use quantum search to find an edge $(a,b) \in E$ among all $\binom{n}{2}$ potential
         edges, using at most $O(n/\sqrt{m'})$ queries.
     (b) Use quantum search to find a node $c \in V$ such that $a,b,c$ is a triangle,
         using at most $(n)$ queries.
  3. Repeat until a triangle is found.
```

Quantum search of an edge $(a,b) \in E$ succeeds after $O(n/\sqrt{m})$ expected queries. Since amplitude amplification forbids any observation in the inner block, we need step 2 to get an estimation of $m$, which determines the number of queries after which step 2(a) will be interrupted.

THEOREM 5.1. *If the graph contains a triangle, then the previous algorithm finds one after $O(n + \sqrt{nm})$ expected queries.*

*Proof.* Suppose the graph contains a triangle. Let an edge be *good* if it is part of a triangle. Then step 2(a) finds one with probability at least $1/2m$. Given this event, step 2(b) finds a triangle with probability at least $1/2$. Now suppose that step 1 found the correct estimation of $m$. Therefore due to the amplitude amplification step 2 succeeds with probability at least $1/2$.

Finally the expected number of repetitions generated by step 3 is constant.

Each iteration costs $O(n + \sqrt{nm'})$ queries, where $m'$ is the random outcome of step 2 with expectation $m$. This establishes the claimed complexity. ∎

## 6. Concluding remarks.

An interesting related problem that is still wide open is the issue of *time-space tradeoffs* for element distinctness. Such tradeoffs have been studied for classical algorithms by Yao [24], Ajtai [2], Beame, Saks, Sun, and Vee [5], and others. In particular, Yao shows that the time-space product of any classical deterministic comparison-based branching program solving element distinctness satisfies $TS \geq \Omega(N^{2-\varepsilon(N)})$, where $\varepsilon(N) = 5/\sqrt{\ln N}$. An upper bound $TS = O((N \log N)^2)$ is achievable classically.

Ignoring logarithmic factors, the quantum algorithm presented here uses time $T = N^{3/4}$ and space $S = N^{1/2}$. An alternative quantum algorithm is to search the space of all $\binom{N}{2}$ $(x,y)$-pairs to try to find a collision. This algorithm has roughly $T = N$ and $S = \log N$. Third, Ambainis's new algorithm has $T = N^{2/3}$ and $S = N^{2/3}$. All these algorithms satisfy $T^2 S \approx N^2$. In fact, for every space bound $S$ less than $N^{2/3}$, one can find an algorithm whose time (or query) complexity $T$ satisfies $T^2 S \approx N^2$. We conjecture that this is close to optimal. Proving this would be very interesting, since no nontrivial quantum time-space tradeoff lower bounds are known for any decision problem (some tradeoffs for multiple-output problems may be found in [17]).

## REFERENCES

[1] S. Aaronson and Y. Shi, *Quantum lower bounds for the collision and the element distinctness problems*, J. ACM, 51 (2004), pp. 595–605.

[2] M. Ajtai, *Determinism versus nondeterminism for linear time RAMs with memory restrictions*, J. Comput. System Sci., 65 (2002), pp. 2–37.

[3] A. Ambainis, *Quantum Walk Algorithm for Element Distinctness*, http://arxiv.org/abs/quant-ph/0311001 (1 Nov 2003).

[4] A. Ambainis, *Quantum Lower Bounds for Collision and Element Distinctness with Small Range*, http://arxiv.org/abs/quant-ph/0305179 (29 May 2003).

[5] P. Beame, M. Saks, X. Sun, and E. Vee, *Super-linear time-space tradeoff lower bounds for randomized computation*, in Proceedings of the 41st IEEE Symposium on Foundations of Computer Science (FOCS), 2000, pp. 169–179.

[6] G. Brassard and P. Høyer, *An exact quantum polynomial-time algorithm for Simon's problem*, in Proceedings of the 5th Israeli Symposium on Theory of Computing and Systems (ISTCS), 1997, pp. 12–23.

[7] G. Brassard, P. Høyer, M. Mosca, and A. Tapp, *Quantum amplitude amplification and estimation*, in Quantum Computation and Quantum Information: A Millennium Volume, Contemp. Math. 305, AMS, Providence, RI, 2002, pp. 53–74.

[8] G. Brassard, P. Høyer, and A. Tapp, *Quantum algorithm for the collision problem*, ACM SIGACT News, 28 (1997), pp. 14–19.

[9] H. Buhrman, R. Cleve, and A. Wigderson, *Quantum vs. classical communication and computation*, in Proceedings of the 30th ACM Symposium on Theory of Computing (STOC), 1998, pp. 63–68.

[10] H. Buhrman, R. Cleve, R. de Wolf, and Ch. Zalka, *Bounds for small-error and zero-error quantum algorithms*, in Proceedings of the 40th IEEE Symposium on Foundations of Computer Science (FOCS), 1999, pp. 358–368.

[11] H. Buhrman, C. Dürr, M. Heiligman, P. Høyer, F. Magniez, M. Santha, and R. de Wolf, *Quantum algorithms for element distinctness*, in Proceedings of the 16th IEEE Conference on Computational Complexity, 2001, pp. 131–137.

[12] C. Dürr, M. Heiligman, P. Høyer, and M. Mhalla, *Quantum query complexity of some graph problems*, in Proceedings of the 31st ICALP, Lecture Notes in Comput. Sci. 3142, Springer-Verlag, New York, 2004, pp. 481–493.

[13] E. Farhi, J. Goldstone, S. Gutmann, and M. Sipser, *Invariant Quantum Algorithms for Insertion into an Ordered List*, http://arxiv.org/abs/quant-ph/9901059 (19 Jan 1999).

[14] D. Grigoriev, *Randomized complexity lower bounds*, in Proceedings of the 30th ACM Symposium on Theory of Computing (STOC), 1998, pp. 219–223.

[15] L. K. Grover, *A fast quantum mechanical algorithm for database search*, in Proceedings of the 28th ACM Symposium on Theory of Computing (STOC), 1996, pp. 212–219.

[16] P. Høyer, J. Neerbek, and Y. Shi, *Quantum complexities of ordered searching, sorting, and element distinctness*, in Proceedings of the 28th ICALP, Lecture Notes in Comput. Sci. 2076, Springer-Verlag, New York, 2001, pp. 346–357.

[17] H. Klauck, R. Špalek, and R. de Wolf, *Quantum and Classical Strong Direct Product Theorems and Optimal Time-Space Tradeoffs*, http://arxiv.org/abs/quant-ph/0402123 (18 Feb 2004).

[18] A. Lubiw and A. Rácz, *A lower bound for the integer element distinctness problem*, Inform. and Comput., 94 (1991), pp. 83–92.

[19] F. Magniez, M. Santha, and M. Szegedy, *An $O(n^{1.3})$ Quantum Algorithm for the Triangle Problem*, http://arxiv.org/abs/quant-ph/0310134 (21 Oct 2003).

[20] M. Mosca, *Quantum searching, counting, and amplitude amplification by eigenvector analysis*, in MFCS Workshop on Randomized Algorithms, 1998, pp. 90–100.

[21] A. Nayak and F. Wu, *The quantum query complexity of approximating the median and related statistics*, in Proceedings of the 31st ACM Symposium on Theory of Computing (STOC), 1999, pp. 384–393.

[22] M. A. Nielsen and I. L. Chuang, *Quantum Computation and Quantum Information*, Cambridge University Press, Cambridge, UK, 2000.

[23] P. W. Shor, *Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer*, SIAM J. Comput., 26 (1997), pp. 1484–1509.

[24] A. C.-C. Yao, *Near-optimal time-space tradeoff for element distinctness*, SIAM J. Comput., 23 (1994), pp. 966–975.

[25] A. C.-C. Yao, *private communication*, Department of Computer Science, Princeton University, Princeton, NJ, 2003.

SIAM J. COMPUT. © 2005 Society for Industrial and Applied Mathematics
Vol. 34, No. 6, pp. 1331–1351

# OUTPUT-SENSITIVE CONSTRUCTION OF THE UNION OF TRIANGLES*

ESTHER EZRA† AND MICHA SHARIR†

**Abstract.** We present an efficient algorithm for the following problem: Given a collection $T = \{\Delta_1, \dots, \Delta_n\}$ of $n$ triangles in the plane, such that there exists a subset $S \subset T$ (unknown to us) of $\xi \ll n$ triangles, such that $\bigcup_{\Delta \in S} \Delta = \bigcup_{\Delta \in T} \Delta$, construct efficiently the union of the triangles in $T$. We show that this problem can be solved in randomized expected time $O(n^{4/3} \log n + n\xi \log^2 n)$, which is subquadratic for $\xi = o(n/\log^2 n)$. In our solution, we use a variant of the method of Brönnimann and Goodrich [*Discrete Comput. Geom.*, 14 (1995), pp. 463–479] for finding a set cover in a set system of finite VC-dimension. We present a detailed implementation of this variant, which makes it run within the asserted time bound. Our approach is fairly general, and we show that it can be extended to compute efficiently the union of simply shaped bodies of constant description complexity in $\mathbb{R}^d$, when the union is determined by a small subset of the bodies.

**Key words.** union of geometric objects, hitting set, finite VC-dimension, random sampling, set cover, $\varepsilon$-net, output sensitivity

**AMS subject classifications.** 52C45, 68U05, 05D99, 51F99, 68W20

**DOI.** 10.1137/S0097539704444245

## 1. Introduction.

Many problems in computational geometry involve the task of constructing the boundary of the union of $n$ geometric objects in the plane or in higher dimensions. Problems of this kind include motion planning [26], where we wish to construct the forbidden portions of the configuration space; hidden surface removal for visibility problems in three dimensions [32]; finding the minimal Hausdorff distance between two sets of points (or of segments) in $\mathbb{R}^2$ [23]; applications in geographic information systems [15]; and many others. In this paper, we focus mainly on the problem of constructing the union of $n$ triangles in $\mathbb{R}^2$, but we also show that our algorithm can be extended to other geometric objects in the plane and in higher dimensions.

Computing the union by constructing the full arrangement of the $n$ input triangles requires $\Theta(n^2)$ time in the worst case, which, in many instances, is wasteful, since the combinatorial complexity of the union boundary might be considerably smaller. Nevertheless, an algorithm for this problem that runs in subquadratic time when the boundary of the union has subquadratic complexity[1] is unlikely to exist, since this problem belongs to the family of 3*SUM-hard* problems [21], which are problems that are very likely to require $\Omega(n^2)$ time in the worst case; see below for more details.

However, subquadratic algorithms exist in several special cases, such as the case of *fat* triangles (namely, every angle of each triangle is at least some constant positive

*Received by the editors June 3, 2004; accepted for publication (in revised form) November 19, 2004; published electronically June 27, 2005. Work on this paper has been supported by NSF grants CCR-97-32101 and CCR-00-98246, by a grant from the U.S.–Israeli Binational Science Foundation, by a grant from the Israel Science Fund, Israeli Academy of Sciences for a Center of Excellence in Geometric Computing at Tel Aviv University, and by the Hermann Minkowski–MINERVA Center for Geometry at Tel Aviv University.

http://www.siam.org/journals/sicomp/34-6/44424.html

†School of Computer Science, Tel Aviv University, Tel Aviv 69978, Israel (estere@post.tau.ac.il, michas@post. tau.ac.il).

[1]This is one variant of *output sensitivity* that one may wish to attain. In this paper we use a different notion of output sensitivity, described later in the introduction.

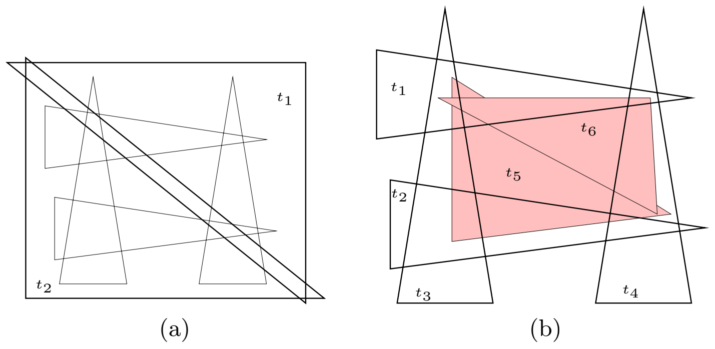


FIG. 1. (a) *An arrangement of six triangles, illustrating the first measure of output sensitivity. The triangles* $t_1$ *and* $t_2$ *cover the entire union, so the output size is* 2. (b) *Illustrating the second measure of output sensitivity. The union boundary is determined only by the triangles* $t_1, \ldots, t_4$*, even though the triangles* $t_5$ *and* $t_6$ *cover the hole created by* $\bigcup_{i\le 4} t_i$*. The output size is* 4 *according to the second measure and* 6 *according to the first.*

angle) or of triangles that arise in the union of Minkowski sums of a fixed convex polygon with a set of pairwise disjoint convex polygons (which is the problem one faces in translational motion planning of a *convex* polygon). In these cases, the union has only linear or near-linear complexity [24, 28, 29], and more efficient algorithms, based on either deterministic divide-and-conquer or on randomized incremental construction, can be devised and are presented in the above-cited papers.

If the input consists of general triangles, then the complexity of the union can be $\Theta(n^2)$ in the worst case. If it happens to be smaller, one can attempt to compute the union by employing the randomized incremental construction (RIC) of Agarwal and Har-Peled [1], whose analysis is based on Mulmuley's *theta series* [32]. Briefly, the algorithm inserts the triangles one at a time in a random order and maintains the union incrementally, updating it after each insertion. As is well known (and discussed in [18]), the RIC algorithm has good performance, even when the size of the arrangement is quadratic, provided that the *depth* $d(v)$ (i.e., the number of input triangles containing $v$ in their interior) of most of the vertices $v$ in the arrangement induced by the $n$ input triangles is large enough. We refer to such vertices as being *deep*. Otherwise, when most of the vertices in the arrangement are *shallow*, the RIC algorithm performs poorly. In this case, one can employ the *disjoint cover* (DC) algorithm proposed in [18], which has good performance in practice. This algorithm also inserts the triangles one at a time, but it computes an insertion order that attempts to cover as many shallow vertices as possible in each insertion step. However, from a theoretical point of view (and in view of certain pathological examples presented in [18]), the DC algorithm can produce $\Omega(n^2)$ vertices of the arrangement, even if the size of the output (i.e., the number of vertices on the boundary of the union) is only linear or constant, and it can be beaten by the RIC algorithm in such cases.

*Output sensitivity.* In this paper we present an efficient algorithm that computes the union in an "output-sensitive" manner. There are two obvious ways to define output sensitivity. The first is to measure the output size in terms of the size of the smallest subset $S \subset T$ that satisfies $\bigcup S = \bigcup T$, where $\bigcup S$ (resp., $\bigcup T$) denotes the union of the triangles in $S$ (resp., in $T$). The second measure is in terms of the size of the smallest subset $S'$ such that $\partial \bigcup T \subseteq \partial \bigcup S'$. See Figure 1 for an illustration of the two measures. Note that if the output size is $\xi$, according to either measure, the actual complexity of the union may be as large as $\Theta(\xi^2)$ (but not larger).

The second measure of output size is likely to be too weak. Indeed, consider the reduction, as presented in [21], of an instance of 3SUM (namely, the problem of determining whether there exist $a \in A$, $b \in B$, $c \in C$ satisfying $a + b + c = 0$ for three given sets $A$, $B$, $C$ of real numbers) to an instance of the problem of determining whether the union of a given set of triangles fully covers the unit square. We can further reduce this latter problem to our problem, as follows. Let $A$ denote an algorithm that efficiently computes the union of $n$ triangles in the plane in terms of the second measure, and let $T_A(n, \xi)$ denote its running time, expressed as a function of $n$ and of the "output size" $\xi$. We assume that $T_A(n, \xi) = o(n^2)$ when $\xi = o(n)$. In order to determine efficiently whether the given triangles fully cover the unit square, we consider only the portions of the triangles that are contained in the unit square and retriangulate them if necessary. In addition, we add four thin and narrow triangles that cover the boundary of the unit square. We now run $A$ on the newly constructed instance. Clearly, there are no holes in the union of the newly created triangles if and only if the original union contains the unit square. In this case, the boundary of the new union consists of only four triangles, and thus $A$ will terminate in a predictable subquadratic time. We thus run $A$. If it terminates within the anticipated (subquadratic) time, we can determine, at no extra cost, whether the union covers the unit square. Otherwise, we stop $A$ and correctly report that the union of the original triangles does not cover the unit square. Hence an efficient output-sensitive solution, under the second measure, would have yielded a subquadratic solution to 3SUM and is thus unlikely to exist.

In contrast, the first measure does lend itself to an efficient output-sensitive solution, which is the main result of this paper.

*Our results.* Specifically, we present an efficient algorithm to construct the boundary of the union of a set $T = \{\Delta_1, \ldots, \Delta_n\}$ of $n$ triangles in the plane, under the assumption that there exists a subset $S \subset T$ of $\xi \ll n$ triangles (unknown to us) such that $\bigcup S = \bigcup T$. We present an algorithm whose running time is $O(n^{4/3} \log n + n\xi \log^2 n)$, which is subquadratic when $\xi = o(n/\log^2 n)$. Our approach is a randomized algorithm based on the method of Brönnimann and Goodrich for finding a set cover or a hitting set in a set system of finite VC-dimension, as presented in [10]. Their method is based on a randomized *natural selection* technique used by Clarkson [12, 13], Littlestone [27], and Welzl [35]; see section 2.1 for a brief review of this method. In our case, the objects are the triangles of $T$, and any point $v$ in the plane defines a set $T_v = \{\Delta \in T \mid v \in int(\Delta)\}$. The collection $\{T_v\}_{v \in \mathbb{R}^2}$ forms a set system for which a *hitting set* $H \subset T$ is a subset satisfying $\bigcup H = \bigcup T$, and thus a minimum-size hitting set is the object that we wish to compute. (It is well known, and easy to verify, that this set system has finite VC-dimension; see below for details.) Note that this set system is the same as the one generated by sampling one point $v$ in the interior of each cell of the arrangement $\mathcal{A}(T)$. In general, the Brönnimann–Goodrich technique is not efficient enough for our purposes, but we use a variant of the algorithm which can be implemented efficiently. Specifically, we apply the algorithm of Brönnimann and Goodrich in an "approximate setting," fine-tuning it (using randomization) so that it constructs a subset $T'$ of $O(\xi \log \xi)$ triangles of $T$, whose union covers the overwhelming majority of the vertices (of positive depth) in the arrangement $\mathcal{A}(T)$. This allows us, with some care, to compute the portion of $\bigcup T$ that lies outside $\bigcup T'$ in an efficient explicit manner. We note that when measuring the expected number of vertices generated by the algorithm, it suffices (and is appropriate) to consider only vertices at positive depth, since vertices at depth 0 are the vertices of the union, and they have to be constructed by any algorithm that computes the union. We call the latter quantity, namely, the number of positive-depth vertices generated by the algorithm, the *residual cost* of the algorithm.

In section 2.1 we briefly recall the algorithm of Brönnimann and Goodrich and present our approximate version of it. Then we derive an upper bound on the expected residual cost of the algorithm in its approximate version. Section 3 describes a detailed implementation of our algorithm. In this implementation, we use generic and simple techniques that can be easily extended to other geometric objects of constant description complexity[2] in the plane and in $\mathbb{R}^d$. These extensions are discussed in section 4. We give concluding remarks and suggestions for further research in section 5.

## 2. The union construction as a set cover problem.

**2.1. An overview of the Brönnimann–Goodrich technique.** A technique for finding a set cover of a set system of finite VC-dimension is described in detail by Brönnimann and Goodrich [10]; for the sake of completeness, we provide a brief overview of this approach, in the context of the union construction problem. Very recently, Even, Rawitz, and Shahar [17] have proposed an alternative technique, based on a linear programming formulation of the problem, that appears to be somewhat simpler and more efficient.

We denote by $V$ the set of vertices of the arrangement $\mathcal{A}(T)$ at positive depth (considering only intersection points of the triangle boundaries and ignoring triangle vertices). A hitting set for the set system induced by $\{T_v\}_{v \in V}$, where $T_v$ consists of all the triangles $\Delta \in T$ that contain $v$ in their interior, is a subset of triangles $H \subset T$ such that $\bigcup H$ covers all the vertices in $V$. It need not necessarily cover $\bigcup T$ entirely, but the pieces left uncovered are easily computable and will be computed in the final stages of the algorithm. Thus we consider the set system $(T, V^*)$, where

$$V^* = \{T_v : v \in V\}.$$

Since this set system is dual to $(V, T)$, which has some finite VC-dimension $d$ (see, e.g., [6]), it follows that the VC-dimension of $(T, V^*)$ is also finite; as a matter of fact, it does not exceed $2^{d+1}$ [8]. As already mentioned, our goal is to find a *hitting set* for $(T, V^*)$, that is, a subset $H \subseteq T$ that has a nonempty intersection with every set $T_v \in V^*$, $v \in V$.

The algorithm of Brönnimann and Goodrich finds a hitting set whose size is $O(h^* \log h^*)$, where $h^*$ is the smallest size of any hitting set. Note that the reported hitting set is actually a *set cover* for the primal set system $(V, T)$, where a set cover, in this case, is a collection $\mathcal{C} \subseteq T$ of triangles whose union covers the entire set $V$. (For technical reasons, the method of Brönnimann and Goodrich computes a set cover via a hitting set of the dual set system, which is why we also work with the dual system; see [10] for further details.) Since, by definition, the size of the optimal cover is assumed to be $\xi$, it follows that the size of the set cover reported by the algorithm is at most $O(\xi \log \xi)$.

We first describe the algorithm of Brönnimann and Goodrich in its "ideal setting," where the entire set $V$ is given, and then show how to modify this setting, so that it suffices to consider only a small subset of vertices.

[2]A set in $\mathbb{R}^d$ is said to have *constant description complexity* if it is a semialgebraic set defined as a Boolean combination of a constant number of polynomial equalities and inequalities of constant maximum degree in a constant number of variables.

The Brönnimann–Goodrich algorithm has two key subroutines: (i) a *net finder* $\mathcal{F}$ for $(T, V^*)$, which is an algorithm that, given a parameter $r \geq 1$ and a weight distribution $w$ on $T$, computes a $(1/r)$-*net* for the weighted system $(T, V^*)$ [6]. A $(1/r)$-net is a subset $N \subseteq T$, which has a nonempty intersection with each set in $V^*$ whose total weight is at least $1/r$ of the total weight of $T$; (ii) a *verifier* $\mathcal{V}$ that, given a subset $H \subseteq T$, either states (correctly) that $H$ is a hitting set, or returns a nonempty "witness" set $T_v \in V^*$, for some $v$, such that $T_v \cap H = \emptyset$. In our context, $\mathcal{V}$ simply has to output a vertex $v \in V$ that is not contained in the interior of $\bigcup H$.

The Brönnimann–Goodrich algorithm then proceeds as follows. We guess the value of $\xi$ (homing in on the right value using an exponential search). We assign weights to the triangles in $T$. Initially, all weights are 1. We then use the net finder $\mathcal{F}$ to construct a $(1/2\xi)$-net $N$ for $(T, V^*)$. If the verifier $\mathcal{V}$ outputs some set $T_v$ that $N$ does not hit, we double the weights of the triangles in $T_v$ and repeat the process with the new weights. As shown in [10], a hitting set is found after at most $4\xi \log (n/\xi)$ iterations.

The problem with this ideal setting is that it requires the construction of all the (positive-depth) vertices of $\mathcal{A}(T)$, which is much too much to ask for, since it can be too expensive ($V$ can be quadratic in the worst case, while $\xi$ can still be very small). Instead, we use a smaller randomly sampled subset $R \subseteq V$ of $r$ elements, whose actual computation is presented in section 3. We then feed the verifier $\mathcal{V}$ with $R$ instead of the entire set $V$. We show that once the verifier $\mathcal{V}$ announces that the subset $H$, reported by the net finder $\mathcal{F}$, covers $R$ (actually, it suffices that $H$ covers most of $R$—see below), the actual number of vertices of $V$ that remain uncovered is relatively small, with high probability. We then compute the uncovered vertices in an explicit manner and thereby complete the construction of $\bigcup T$.

**2.2. A subquadratic residual cost via sampling.** We begin the analysis of our implementation of the Brönnimann–Goodrich technique with the following lemma, which provides a lower bound for the size of the sample $R$, which is sufficient to guarantee the property asserted at the end of the preceding subsection.

In what follows, we say that an event occurs *with overwhelming probability* (or w.o.p., for short) if the probability that it does not occur is at most $\frac{1}{n^c}$ for some constant $c \geq 1$.

LEMMA 2.1. *Let $T = \{\Delta_1, \ldots, \Delta_n\}$ be a given collection of $n$ triangles in the plane, let $V$ denote the set of vertices of the arrangement $\mathcal{A}(T)$ at positive depth, let $\kappa$ denote the size of $V$, and suppose that there are only $\xi$ triangles of $T$ whose union is equal to $\bigcup T$. Let $S \subseteq T$ denote a subset of triangles, and let $R \subseteq V$ be a random sample of $r = \Omega(t \log n)$ positive-depth vertices, sampled after $S$ has been fixed for some prespecified parameter $t \geq 1$ and with a sufficiently large constant of proportionality. If $S$ covers all but $r_S < r$ vertices of $R$, then, w.o.p., the actual number $\kappa_S$ of vertices of $V$ that are not covered by the elements of $S$ satisfies*

$$\kappa_S \leq \max\left\{\frac{\kappa}{t}, \beta\frac{\kappa}{r} r_S\right\} \tag{2.1}$$

*for some absolute constant $\beta > 1$.*

*Proof.* For simplicity of exposition, we present the analysis under the model where $R$ is obtained by drawing each point of $V$ independently with probability $p = \frac{r}{\kappa}$. Nevertheless, the assertion of the lemma also holds for other models of sampling $R$, in particular, for the model we use in the actual implementation of the algorithm; see section 3 and Appendix A for details. Since each point in $V \setminus \bigcup S$ is chosen independently with probability $\frac{r}{\kappa}$, the expected number of vertices of $R$ that are not covered by $S$ is $\frac{r}{\kappa}\kappa_S$.

It suffices to consider the case $\kappa_S > \frac{\kappa}{t}$, for otherwise (2.1) clearly holds.

Since $R$ is sampled *after* $S$ has been fixed, the number $r_S$ of vertices of $R$ that are not covered by $\bigcup S$ is a random variable, which can be expressed as the sum of $\kappa_S$ mutually independent indicator variables, $X_1, \ldots, X_{\kappa_S}$, each satisfying

$$Pr[X_i = 1] = p, \qquad Pr[X_i = 0] = 1 - p \quad \text{for } i = 1, \ldots, \kappa_S.$$

Fix a parameter $r_0 > 0$ and consider the event

$$A_S : \quad r_S - \frac{r}{\kappa}\kappa_S < -r_0.$$

Using a large deviation bound given in [6, Theorem A.13], it follows that

$$Pr[A_S] < e^{-\frac{{r_0}^2}{2\frac{r}{\kappa}\kappa_S}}. \tag{2.2}$$

Putting $r_0 = \sqrt{2c_0\frac{r}{\kappa}\kappa_S \log n}$ for some constant $c_0 \geq 1$, (2.2) implies that the probability that the event $A_S$ does not occur is at most $\frac{1}{n^{c_0}}$. Hence, w.o.p.,

$$r_S - \frac{r}{\kappa}\kappa_S \geq -\sqrt{2c_0\frac{r}{\kappa}\kappa_S \log n},$$

or

$$r_S \geq \sqrt{\frac{r}{\kappa}\kappa_S}\left[\sqrt{\frac{r}{\kappa}\kappa_S} - \sqrt{2c_0 \log n}\right].$$

Since we have assumed that $\kappa_S > \frac{\kappa}{t}$ and that $r = \Omega(t \log n)$, with a sufficiently large constant of proportionality, it follows that, w.o.p.,

$$\sqrt{\frac{r}{\kappa}\kappa_S} - \sqrt{2c_0 \log n} > \alpha\sqrt{\frac{r}{\kappa}\kappa_S} \tag{2.3}$$

for some absolute constant $0 < \alpha < 1$, which implies that

$$\kappa_S \leq \frac{\kappa}{\alpha r} r_S,$$

and thus the lemma follows. □

*Remarks.* (1) Note that Lemma 2.1, as well as its variant discussed in Appendix A, deals with abstract sets, and does not exploit any special property of vertices in arrangements of triangles. We will therefore be able to use the lemma, more or less verbatim, in the extensions presented in section 4.

(2) We reemphasize that Lemma 2.1 relies on the assumption that $R$ is sampled *after* $S$ has been chosen (in our implementation, this choice will also be random). In particular, for the lemma to be applicable at each iteration of the Brönnimann–Goodrich algorithm, $R$ should be redrawn from scratch before applying the verifier $\mathcal{V}$. (See section 3 for further details.)

Lemma 2.1 implies that if the triangles in $S$ cover all but at most $\frac{r}{t}$ of the elements of $R$ (and thus $r_S = O\left(\frac{r}{t}\right)$), then, w.o.p., $\kappa_S \leq \frac{\kappa}{t}$. We thus construct the union of the input triangles in two steps: in the first we find a set $H$ of $O(\xi \log \xi)$ triangles that covers all but at most $\frac{\kappa}{t}$ vertices of $V$ and compute the union $\bigcup H$, and in the second step we handle efficiently all the remaining vertices of $V$ that $H$ does not cover; see below for details. It thus follows that the overall expected number of positive-depth vertices generated by the algorithm is $O(\xi^2 \log^2 \xi)$ (which is the number of vertices of the arrangement of the triangles in $H$) in the first part, and at most $\frac{\kappa}{t}$ in the second part.

We have established the following theorem.

THEOREM 2.2. *Let $T = \{\Delta_1, \ldots, \Delta_n\}$ be a given collection of $n$ triangles in the plane, and assume that there exists a subset $H \subset T$ of $\xi \ll n$ triangles (unknown to us) such that $\bigcup H = \bigcup T$. Let $V$, $\kappa$, and $t$ be as in Lemma* 2.1*. Then one can implement the Brönnimann–Goodrich algorithm, so that its residual cost is $O(\xi^2 \log^2 \xi + \frac{\kappa}{t})$, w.o.p. In particular, for $t = \max\{\frac{\kappa}{\xi^2}, 1\}$ the residual cost is $O(\xi^2 \log^2 \xi)$.*

*Discussion.* Clearly, if our only concern is to have the algorithm generate as few positive-depth vertices as possible, we should choose $t$ as large as possible, thereby making $R$ larger and the set of vertices of $V$ not covered by $H$ smaller. For example, as noted, if we choose $t = \max\{\frac{\kappa}{\xi^2}, 1\}$, then the residual cost of the algorithm is at most $O(\xi^2 \log^2 \xi)$, w.o.p. Since there are only $\xi$ triangles that define the union, the combinatorial complexity of the boundary of the union is only $O(\xi^2)$. This implies that for the above choice of $t$, the overall number of vertices that the algorithm generates is $O(\xi^2 \log^2 \xi)$, which is subquadratic for $\xi = o(n/\log n)$. However, if we are concerned with the actual running time, large values of $t$ will slow down the algorithm, because sampling the sets $R$ will be more expensive. Hence, in the actual implementation of the algorithm, presented in section 3 below, we will choose a smaller value for $t$ in order to optimize the bound on the actual running time of the algorithm. This will also affect the bound on the residual cost.

We also note that the bound $O(\xi^2 \log^2 \xi)$ on the complexity of the union of the triangles computed in the first part of the algorithm may be too pessimistic in practice. If the complexity of the union $\bigcup H$ turns out to be smaller, the residual cost will be smaller too.

## 3. Implementation of the algorithm.

The actual cost of the algorithm depends on the cost of several support routines (in addition to the cost of the actual generation of positive-depth vertices), such as (i) constructing the random samples $R$; (ii) finding a $(1/2\xi)$-net for the set system $(T, V^*)$; (iii) implementing the verifier $\mathcal{V}$, which in our case is an algorithm that efficiently decides whether a given subset $S$ of triangles covers (most of the elements of) another given subset $R$ of positive-depth vertices; and (iv) the actual construction of the union of the input triangles after an approximate hitting set has been found. We present here an implementation that uses generic and simple techniques and yields a subquadratic output-sensitive algorithm for constructing the union.

In the following description, we denote by $h$ the size of the set $H$ computed in the first stage of the algorithm.

**Sampling *R*.** The task at hand is to construct, at each iteration of the algorithm, a random sample of (an expected number of) $r = ct \log n$ positive-depth vertices of $\mathcal{A}(T)$, for appropriate values of the parameter $t$ and the constant $c$. (As already mentioned and to be discussed below, we have to draw a new subset $R$ in each iteration of the algorithm in order to eliminate any dependence between the present subset of triangles reported by the net finder $\mathcal{F}$ and the (current) sample $R$.)

We sample $R$ using the following simple approach. Suppose that we have a guess for the values of $\xi$ and $\kappa$ (see below for details concerning these guesses). Let $\kappa^*$ denote the number of vertices on the boundary of $\bigcup T$. If $\kappa = O(\kappa^*)$, then the entire arrangement has only $O(\kappa^*) = O(\xi^2)$ vertices and can thus be constructed in time $O(n \log n + \xi^2)$, using any of the standard techniques [32]. We may thus assume that $\kappa > \beta\kappa^*$ for some absolute constant $\beta > 1$. We also may assume that $\kappa > \beta \max\{\xi^2, n^{4/3}\}$ for the same constant. Otherwise, we construct the entire arrangement in time $O((n + \xi^2 + n^{4/3}) \log n) = O((\xi^2 + n^{4/3}) \log n)$.

We now perform $\frac{9c'r\binom{n}{2}}{\kappa}$ sampling steps, where in each step we choose, uniformly and independently, a pair of edges of distinct triangles in $T$ for an appropriate constant $c' > 1$ (we first select a random pair of triangles and then randomly choose a pair of triangle edges out of the nine triangle edge pairs induced by the two chosen triangles). Clearly, a real vertex of the arrangement $\mathcal{A}(T)$ (that is formed by a pair of intersecting triangle edges) is chosen in a single step with probability $\frac{\kappa+\kappa^*}{9\binom{n}{2}}$, and thus the expectation of the number $r'$ of pairs of edges that actually intersect is

$$\frac{\kappa + \kappa^*}{9\binom{n}{2}} \cdot \frac{9c'r\binom{n}{2}}{\kappa} = \Theta(r).$$

Applying the same deviation bound used in Lemma 2.1, it can be shown that, w.o.p., the actual number of such pairs satisfies

$$r' \geq \mathbf{E}(r') - \sqrt{\gamma \frac{\kappa}{9\binom{n}{2}} \frac{9c'r\binom{n}{2}}{\kappa} \log n} = \mathbf{E}(r') - \sqrt{\gamma c' r \log n}$$

for some constant $\gamma \geq 1$. Since $\sqrt{\gamma c' r \log n} = o(r)$ (by the choice of $\gamma$, $c'$, and $r$), there is a constant $0 < \alpha < 1$, which can be made arbitrarily small (for a proper choice of $\gamma$) such that, w.o.p.,

$$r' \geq (1 - \alpha)\mathbf{E}(r') = \Theta(r)$$

for a sufficiently large constant of proportionality that depends on $c'$ and $\gamma$.

Not all sampled vertices have positive depth. However, since $\kappa > \beta\kappa^*$, the overwhelming majority of the sampled vertices will have positive depth. By choosing $c'$ to be sufficiently large, at least $r$ of these vertices will have positive depth, w.o.p.

**Implementing a net finder $\mathcal{F}$ and a verifier $\mathcal{V}$.** As already described in the preceding section, we assign weights to the elements of $T$ (initially, each triangle gets the weight 1) and use a net finder $\mathcal{F}$ to construct a $(1/2\xi)$-net for the weighted dual system $(T, V^*)$. We then apply the verifier $\mathcal{V}$ in order to decide whether $H$ covers (most of the elements of the newly resampled subset) $R$. If it does, the first part of the algorithm terminates, and we proceed to the actual construction of the union; otherwise, $\mathcal{V}$ returns a particular witness subset $T_v \in V^*$, for some $v \in R$, such that $T_v \cap H = \emptyset$. We then double the weights of the triangles in $T_v$, construct a new $(1/2\xi)$-net and a new sample $R$, and repeat this process until we find a subset of triangles that covers all but at most $\frac{r}{t}$ elements of $R$. The analysis in [10] can be modified to show that the number of iterations that this algorithm performs is $O(\xi \log(n/\xi))$. Indeed, as long as there exists some vertex of the new sample $R$ that is not covered by the set $H$ constructed by $\mathcal{F}$, we keep on doubling the weights of the triangles covering this vertex, and, according to the analysis in [10], the overall number of such iterations does not exceed $4\xi \log(n/\xi)$.

We start with the description of the net finder $\mathcal{F}$. We use a simple method, presented by Matoušek [30] and briefly reviewed in [10], for reducing the weighted case to the unweighted one. In this method, we scale all weights of the triangles in $T$ such that the sum $w(T)$ of the weights of all the elements of $T$ satisfies $w(T) = n$. We then take $\lfloor w(\Delta) + 1 \rfloor$ copies of each element $\Delta \in T$ (where $w(\Delta)$ is the scaled weight of $\Delta$). Note that the multiset $T'$ that we have constructed contains all the elements of $T$ and has at most $2n$ elements. It is shown in [30] that an $\varepsilon$-net for (the unweighted set) $T'$ is also an $\varepsilon$-net for the weighted set $T$. Finding a $(1/2\xi)$-net for $T'$ can be done by drawing $O(\xi \log \xi)$ random elements of $T'$. As shown, e.g., in [6], an appropriate choice of the constant of proportionality ensures that such a random sample is a $(1/2\xi)$-net, w.o.p. Clearly, creating the multiset $T'$ takes $O(n)$ time, and drawing $O(\xi \log \xi)$ random elements of $T'$ takes an additional time of $O(\xi \log \xi)$. Thus the overall running time of the net finder is $O(n)$, for a total time of $O(n\xi \log(n/\xi))$ over all iterations of the algorithm. Note that if the random sample is not a $(1/2\xi)$-net (which may happen with an overwhelmingly small probability for any $\xi \geq \log n$; see, e.g., [22]), the number of iterations of the algorithm may exceed $4\xi \log(n/\xi)$, and, in this case, we may stop the whole process and restart it from scratch. The fact that the process fails with an overwhelmingly small probability ensures that, w.o.p., the number of such trials is not larger than some constant. (When $\xi < \log n$, the number of trials that guarantees success, w.o.p., is at most $O(\log n)$. However, this does not affect the asymptotic running time of the algorithm; see section 3 for the specific bound on the running time of the algorithm.)

In the implementation of the verifier $\mathcal{V}$, we use brute force and iterate over all the vertices of $R$ and the triangles of $H$ in $O(r\xi \log \xi)$ time, to determine whether there exists a vertex in $R$ that is not covered by the triangles of $H$. We denote the set of all such vertices of $R$ by $R_H$. Suppose $R_H$ contains at least $\frac{r}{t}$ vertices (otherwise, the first part of the algorithm terminates). Rather than just picking any $v \in R_H$, we sample a random vertex $v$ from $R_H$, obtain, by brute force, the set $T_v$ of all triangles in $T$ that contain $v$ in their interior (clearly, $T_v \cap H = \emptyset$), and if $T_v \neq \emptyset$, double their weights. The reason for sampling is that $R$ may in general also contain zero-depth vertices, and $T_v$ will be empty for such vertices $v$. To accommodate this case, we use the sampling technique and stop when we find a positive-depth vertex in $R_H$. Since (i) $|R_H| \geq \frac{r}{t} = \Omega(\log n)$, (ii) $\kappa > \beta\kappa^*$, and (iii) $R$ is sampled after $H$ has been constructed, it follows that a constant positive fraction of the elements of $R_H$ have positive depth, and that, w.o.p., such an element will be found after at most $O(\log n)$ samplings. Hence, w.o.p., the total cost of this substep is $O(n \log n)$. Since we repeat this procedure for $O(\xi \log(n/\xi))$ steps, the overall cost of this stage is, w.o.p.,

$$O(\xi \log(n/\xi)(r\xi \log \xi + n \log n)) = O(r\xi^2 \log \xi \log(n/\xi) + n\xi \log(n/\xi) \log n),$$

and this bounds the overall running time, for both the net finder $\mathcal{F}$ and the verifier $\mathcal{V}$, over all iterations of the first part of the algorithm.

**The actual construction of the union.** The implementation of the actual construction of the union proceeds through two stages. We first construct the union of the triangles in the set $H$ and then compute the portion of $\mathcal{A}(T)$ outside this union. As argued earlier, this portion contains, w.o.p., at most $\frac{\kappa}{t}$ positive-depth vertices of $\mathcal{A}(T)$.

We first construct the union of the $h$ triangles of $H$ in $O(h^2) = O(\xi^2 \log^2 \xi)$ time (using, e.g., randomized incremental construction [32]). Next, we efficiently find the intersections of the boundary of each of the remaining triangles $\Delta$ with the boundary

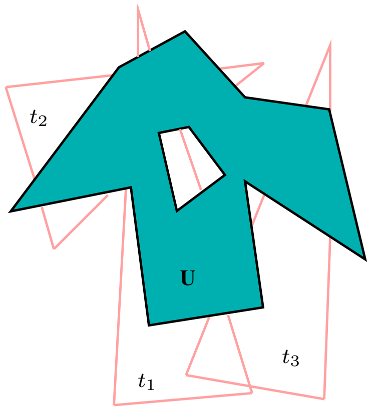


FIG. 2. *The second stage of the actual construction of the union. $U$ denotes the union of the $h$ triangles in the hitting set $H$ and $t_1, t_2$, and $t_3$ denote the remaining triangles to be inserted into the union. Only the portions of $t_1, t_2$, and $t_3$ that lie outside $U$ are relevant.*

of $\bigcup H$ in order to collect all the portions of $\partial\Delta$ lying outside $\bigcup H$. We denote the set of all such portions, over all the remaining triangles, by $\mathcal{C}$. (See Figure 2 for an illustration.)

In order to find those portions efficiently, we use the algorithm of Bentley and Ottmann [9] (see also [16]) for reporting all $k$ intersections in a set of $n$ simply shaped Jordan arcs in $O(n \log n + k \log n)$ time. We partition the set of the remaining triangles into $\lceil \frac{n}{\xi \log \xi} \rceil$ subsets, each containing $O(\xi \log \xi)$ triangles. We denote the collection of all these subsets by $\mathcal{S} = \{S_1, \ldots, S_{\lceil \frac{n}{\xi \log \xi} \rceil}\}$. Next, we compute, for every subset $S \in \mathcal{S}$, the arrangement $\mathcal{A}(S)$ induced by the triangles in $S$ and then run the Bentley–Ottmann algorithm on the combined collection of the edges of $\mathcal{A}(S)$ and the $O(h^2)$ edges of $\bigcup H$. Since the edges of $\mathcal{A}(S)$ are pairwise openly disjoint, as are the edges of $\bigcup H$, the algorithm will report only intersections between the boundary of $\bigcup H$ and the remaining triangles. Since the overall number of such intersections over all subsets in $\mathcal{S}$ is at most $\frac{\kappa}{t}$, the overall cost of reporting all intersections is

$$O\left(\left(\frac{n}{\xi \log \xi} \cdot \xi^2 \log^2 \xi\right) \log n + \frac{\kappa}{t} \log n\right) = O\left(n\xi \log \xi \log n + \frac{\kappa}{t} \log n\right).$$

Next, we trim the edges of the remaining triangles to their portions outside $\bigcup H$ and then construct the entire union using another line-sweeping procedure on these exterior edge portions and the boundary edges of $\bigcup H$ [9]. Since there are at most $\frac{\kappa}{t}$ positive-depth vertices that are constructed during this process, the algorithm takes $O((n + \xi^2 \log^2 \xi + \frac{\kappa}{t}) \log n)$ time.

This completes the detailed description of our algorithm, which is summarized in the following procedure for which $\xi$ is an input parameter. Since $\xi$ is not known a priori, we run this procedure with the values $\xi = 1, 2, 4, \ldots, 2^i, \ldots$ (where $i < \log n$), thereby guaranteeing a constant approximation of the actual value of $\xi$. The choice of $r$ (that is, of the parameter $t$) in this procedure will be specified later.

**Procedure** CONSTRUCTUNION($T$, $\xi$)

```
1. Construct $\bigcup T$ by a line-sweeping procedure on the triangles in $T$. Stop the
   procedure as soon as it constructs more than $\max\{\xi^2, n^{4/3}\}$ vertices.
   If it terminates **goto** 16.
```

```
2.    Initialize all weights of the triangles in $T$ to 1.
3.    **repeat**
4.      $H \leftarrow (1/2\xi)$-net of size $O(\xi \log \xi)$ for the weighted system $(T, V^*)$.
5.      Construct a new random sample $R$ of $r$ vertices out of the vertices of $\mathcal{A}(T)$.
6.      Apply the verifier $\mathcal{V}$ to $H$ and $R$.
7.      **if** $H$ covers all but at most $\frac{r}{t}$ vertices of $R$ **goto** 11.
8.      **else**
9.        Double the weights of all the triangles in the subset $T_v$ reported by $\mathcal{V}$.
10.   **endrepeat**
11.   Construct the union of the triangles in $H$.
12.   Partition $T$ into subsets $S_1, \ldots, S_{\lceil \frac{n}{\xi \log \xi} \rceil}$ of size $O(\xi \log \xi)$ each.
13.   For each $S_i$, compute $\mathcal{A}(S_i)$ and find all intersections between its edges and
      $\partial \bigcup H$, using a line-sweeping procedure.
14.   Trim the edges of the remaining triangles to their portions outside $\bigcup H$.
      Denote the set of the resulting segments by $\mathcal{C}$.
15.   Construct $\bigcup T$ by a line-sweeping procedure on $\mathcal{C}$ and the boundary edges of
      $\bigcup H$.
16. **end**
```

We substitute $r = ct \log n$ for some absolute constant $c$ and for the parameter $t$ that we still need to fix. Since the size $h$ of $H$ is $O(\xi \log \xi)$ and since the algorithm terminates after $O(\xi \log (n/\xi))$ iterations, the overall cost of the algorithm (including the exponential search of the actual value of $\xi$) is

$$\min \left\{ \begin{array}{l} O((n+\kappa)\log n), \\ O\Big(\frac{n^2}{\kappa} r\xi \log(n/\xi) + n\xi \log(n/\xi)\log n + hr\xi \log(n/\xi) \\ \qquad + nh \log n + \frac{\kappa}{t}\log n + h^2 \log n\Big) \end{array} \right\}$$

$$= \min \left\{ \begin{array}{l} O((n+\kappa)\log n), \\ O\Big(\frac{n^2}{\kappa} t\xi \log n \log(n/\xi) + n\xi(\log(n/\xi) + \log \xi)\log n \\ \qquad + \xi^2 t \log \xi \log n \log(n/\xi) + \frac{\kappa}{t}\log n\Big) \end{array} \right\}.$$

Choosing

$$t = \max\left\{ \frac{\sqrt{\kappa}}{\xi \log n}, 1 \right\},$$

the running time bound becomes

$$\min\left\{ O((n+\kappa)\log n), O\left( \frac{n^2}{\sqrt{\kappa}} \log(n/\xi) + \xi\sqrt{\kappa}\log^2 n + n\xi(\log(n/\xi) + \log \xi)\log n \right) \right\}.$$

Since $\kappa = O(n^2)$ and $\xi \le n$, this is upper bounded by

$$\min\left\{ O((n+\kappa)\log n), O\left( \frac{n^2}{\sqrt{\kappa}}\log n + n\xi \log^2 n \right) \right\}.$$

The two terms involving $\kappa$ are equal when $\kappa = n^{4/3}$. Hence the running time is always bounded by $O(n^{4/3}\log n + n\xi \log^2 n)$.

We have established the following theorem.

Theorem 3.1. *Let $T$ be a set of $n$ triangles in the plane whose union is equal to the union of an unknown subset of $\xi \ll n$ triangles. Then the union can be constructed in randomized expected time $O(n^{4/3} \log n + n\xi \log^2 n)$, which is subquadratic for any $\xi = o(\frac{n}{\log^2 n})$.*

## 4. Extensions.

In this section we show how to extend our algorithm to compute the union of other planar shapes, as well as unions of simply shaped bodies in three and higher dimensions.

The analysis of the algorithm of [10] holds for any range space of finite VC-dimension. Consider an input set $S$ of bodies in $\mathbb{R}^d$, and let $V$ denote the set of positive-depth vertices of $\mathcal{A}(S)$. It is well known that the range space $(S, V^*)$ has finite VC-dimension if the objects have constant description complexity. This can be shown, for instance, by the linearization technique (see, e.g., [31]). In this case, the number of vertices that the objects in the set $H$, reported by the net finder $\mathcal{F}$, can generate, among themselves, is $O(\xi^d \log^d \xi)$. In addition, Lemma 2.1 continues to hold in this case, since it does not make any assumptions on the input shapes. It thus follows that Theorem 2.2 can be easily extended to bodies in $\mathbb{R}^d$ of constant description complexity, and that the residual cost of the algorithm, in this case, is $O(\xi^d \log^d \xi + \frac{\kappa}{t})$, w.o.p.

The actual implementation of the various stages of the algorithm can also be easily extended to bodies in $\mathbb{R}^d$ of constant description complexity. We begin with the planar case, and then discuss in section 4.1 the extension to higher dimensions.

In the case of simply shaped planar regions, we apply similar subroutines, which run within the same time bounds as stated in section 3. In the sampling procedure, each pair of region boundaries intersect in a constant number of points, and we collect all these intersections to form $R$. Since our system has finite VC-dimension, we can construct a $(1/2\xi)$-net for this system in much the same way as in section 3. In addition, the verifier $\mathcal{V}$ can still detect whether a given vertex $v$ is contained in the interior of another given region in $O(1)$ time, and thus these two subroutines will run within the same asymptotic time bounds as in the case of triangles. (In fact, these properties hold for bodies of constant description complexity in higher dimensions as well, and thus the net finder $\mathcal{F}$ and the verifier $\mathcal{V}$ will run within the same asymptotic time bounds in these cases too.) In the actual construction of the union, we use the algorithm of Bentley and Ottmann [9], which can be applied for any set of Jordan arcs of constant description complexity, with the same asymptotic time bound, as stated in section 3.

We can thus easily derive the following theorem.

Theorem 4.1. *Let $S$ be a set of $n$ planar regions of constant description complexity, whose union is equal to the union of an unknown subset of $\xi \ll n$ regions. Then the union can be constructed in randomized expected time $O(n^{4/3} \log n + n\xi \log^2 n)$, which is subquadratic for any $\xi = o(\frac{n}{\log^2 n})$.*

### 4.1. The union of simply shaped bodies in $\mathbb{R}^d$.

We begin with the extension of our algorithm to the case of bodies of constant description complexity in three dimensions and then describe the generalization to higher dimensions.

In 3-space, we may assume in the sampling procedure that $\kappa > \beta \max\{\xi^3, n^2\}$ for some absolute constant $\beta > 1$. Otherwise, we construct the union in time $O((n^2 + \xi^3) \log n)$, as follows. We fix a body $B \in S$ and intersect its boundary $F$ with each object $B' \in S \setminus \{B\}$. We obtain a collection of $n-1$ Jordan regions of constant description complexity on $F$. The complement of their union is the portion of $F$ that appears on $\partial \bigcup S$. Computing this complement can be done in time $O(n \log n + \kappa_B \log n)$, where $\kappa_B$ is the number of vertices of $\mathcal{A}(S)$ that lie on $F$, using an appropriate variant of the line-sweeping algorithm of Bentley and Ottmann [9]. Repeating this procedure for each boundary $F$, the total cost is $O((n^2 + \kappa) \log n) = O((n^2 + \xi^3) \log n)$, as claimed.

The main part of the algorithm then proceeds in much the same way as before. For example, when we construct a sample $R$ of vertices, we perform, in analogy with the two-dimensional procedure, $\frac{c' r \binom{n}{3}}{\kappa}$ sampling steps, for an appropriate constant $c' > 1$, where in each step we choose, uniformly and independently, a triple of distinct input bodies in $S$ and collect all resulting boundary intersections to form $R$. An analysis similar to that described in section 3 shows that with an appropriate choice of the constant $c'$, at least $r$ of the chosen triples generate real vertices that have positive depth, w.o.p.

As noted above, the net finder $\mathcal{F}$ and the verifier $\mathcal{V}$ can be implemented in a manner similar to that described in section 3 and run within the same asymptotic time bounds (and this holds in higher dimensions as well). It follows that, choosing $t = \max\{\frac{\sqrt{\kappa}}{\xi \log n}, 1\}$, the first part of the algorithm computes a subset $H$ of $S$ of size $h = O(\xi \log \xi)$, in time $O(r\xi^2 \log \xi \log(n/\xi) + n\xi \log(n/\xi) \log n)$, such that at most $\frac{\kappa}{t}$ positive-depth vertices of $\mathcal{A}(S)$ lie outside the (interior of the) union $\bigcup H$.

After constructing $\bigcup H$, we need to compute all the intersections between the remaining bodies and the boundary of $\bigcup H$. This is done as follows. For each body $B \in S$ (particularly, $B$ may belong to $H$), we take its boundary $F$ and compute the set of its exposed portions that lie outside $\bigcup H \setminus \{B\}$. This is done by constructing the intersections $B'_F = B' \bigcap F$ for each $B' \in H \setminus \{B\}$, and then we compute the complement of their union within $F$. Since the regions $B'_F$ are bounded by curves of constant description complexity, their arrangement has $O(h^2)$ complexity and can be constructed in $O(h^2 \log n)$ time. We denote by $E_F$ the set of edges of the arrangement that appear on the boundary of the union of the regions $B'_F$. Clearly $|E_F| = O(h^2)$. We then intersect $F$ with all the remaining $n - h$ input bodies, obtaining a set of curves $\mathcal{S}_F$ bounding the intersection regions. Our goal is to find the portions of the curves in $\mathcal{S}_F$ that are not contained in the interior of $\bigcup H$; see Figure 3 for an illustration. We first report the intersections between the curves in $\mathcal{S}_F$ and $E_F$ in $O(nh \log n + I_F \log n)$ time, where $I_F$ is the number of such intersections, in a manner similar to that described in the two-dimensional case. Since the overall number of these intersections, over all boundaries $F$, is less than $\frac{\kappa}{t}$, the overall time needed to report all these intersections, over all these boundaries, is

$$O\left(n^2 h \log n + \frac{\kappa}{t} \log n\right).$$

We now trim, on each boundary $F$, the edges of the cross sections of the remaining input bodies, to their portions outside $\bigcup H$, and continue in a similar manner to that described in the two-dimensional case; that is, we run a line-sweeping procedure on these portions and the curves in $E_F$. This constructs the entire two-dimensional arrangements that these portions induce, from which the complete union boundary is easy to extract. The running time of this procedure over all boundaries $F$ is $O((n^2 + nh^2 + \frac{\kappa}{t}) \log n)$.

The overall running time of the algorithm in this case is thus

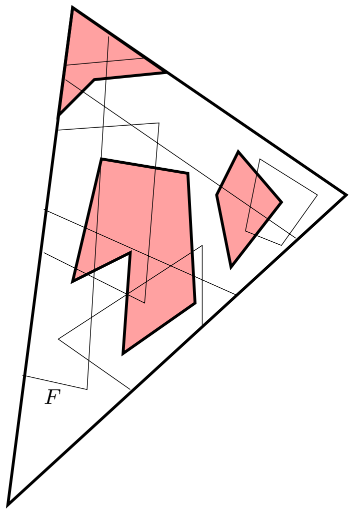


FIG. 3. *The case where the input bodies are simplices in three dimensions. The facet $F$ belongs to one of the $h$ simplices in $H$. The thick lines are the boundaries of $\bigcup H$ on $F$. The thin lines are the intersections of the $n-h$ remaining simplex boundaries with $F$. The intersections appearing in the shaded regions lie in the interior of the union of the $n$ simplices and need not be computed explicitly.*

$$\min\left\{\begin{array}{l} O((n^2+\kappa)\log n), \\ O\left(\frac{n^3}{\kappa}r\xi\log(n/\xi)+n\xi\log(n/\xi)\log n+hr\xi\log(n/\xi)+n^2h\log n+\frac{\kappa}{t}\log n\right)\end{array}\right\}$$

$$=\min\left\{\begin{array}{l} O((n^2+\kappa)\log n), \\ O\left(\frac{n^3}{\kappa}t\xi\log n\log(n/\xi)+\xi^2t\log\xi\log n\log(n/\xi)+n^2\xi\log\xi\log n+\frac{\kappa}{t}\log n\right)\end{array}\right\}.$$

Choosing, as above,

$$t=\max\left\{\frac{\sqrt{\kappa}}{\xi\log n},1\right\},$$

the running time bound becomes

$$\min\left\{O((n^2+\kappa)\log n), O\left(\frac{n^3}{\sqrt{\kappa}}\log(n/\xi)+\xi\sqrt{\kappa}\log^2 n+n^2\xi\log\xi\log n\right)\right\}.$$

Since $\kappa=O(n^3)$ and $\xi\le n$, this is upper bounded by

$$\min\left\{O((n^2+\kappa)\log n), O\left(\frac{n^3}{\sqrt{\kappa}}\log n+n^2\xi\log^2 n\right)\right\}.$$

The two terms involving $\kappa$ are equal when $\kappa=n^2$. Hence the running time is always bounded by

$$O(n^2\log n+n^2\xi\log^2 n)=O(n^2\xi\log^2 n),$$

which is subcubic for $\xi=o(\frac{n}{\log^2 n})$.

Consider next the union problem in $d \geq 4$ dimensions. Let $\mathcal{B}$ be a set of $n$ bodies of constant description complexity in $\mathbb{R}^d$, and let $\mathcal{S} \subset \mathcal{B}$ be the (unknown) subset of $\xi$ bodies whose union is equal to $\bigcup \mathcal{B}$. We compute the union by recursing on the dimension. That is, we fix a body $B \in \mathcal{B}$, take its boundary $F$, and intersect it with each body $B' \in \mathcal{B} \setminus \{B\}$. We then compute the union of these intersection bodies and construct its component within $F$. The union of all these components over all boundaries $F$ yields the boundary of $\partial \mathcal{B}$. Note that if $B \in \mathcal{B} \setminus \mathcal{S}$, then the union of the intersection bodies along $\partial B$ covers the entire boundary of $B$. In fact, the union of the intersections with the bodies of $\mathcal{S}$ already covers the boundary. Similarly, if $B \in \mathcal{S}$, then the union of the intersection bodies along $\partial B$ is equal to the union of the intersections with the bodies of $\mathcal{S}$. In either case, with an appropriate parametrization of the boundaries, we obtain $n$ $(d-1)$-dimensional instances of the union construction problem, each with output size $\leq \xi$, according to our measure. We thus compute these $(d-1)$-dimensional unions recursively and stop the recursion when $d = 3$. This leads to an overall algorithm that runs in randomized expected time $O(n^{d-1}\xi \log^2 n)$. That is, we have the following theorem.

THEOREM 4.2. *Let $S$ be a set of $n$ bodies of constant description complexity in $\mathbb{R}^d$, whose union is equal to the union of an unknown subset of $\xi \ll n$ bodies. Then the union can be constructed in randomized expected time $O(n^{d-1}\xi \log^2 n)$, which is asymptotically smaller than $n^d$ for any $\xi = o(\frac{n}{\log^2 n})$.*

## 5. Concluding remarks.

We have presented an output-sensitive algorithm for the problem of constructing efficiently the union of $n$ triangles in the plane, whose running time is expressed in terms of the smallest size $\xi$ of an unknown subset of the triangles whose union is equal to the union of the entire set. We have used a variant of the technique of Brönnimann and Goodrich [10] for finding an approximate set cover in a set system of finite VC-dimension. We have also presented a detailed and fairly generic implementation of this method, showing that the above problem can be solved in randomized expected time $O(n^{4/3} \log n + n\xi \log^2 n)$, which is subquadratic for $\xi = o(\frac{n}{\log^2 n})$. Derandomization of our implementation seems nontrivial, and an open problem that thus arises is to make the worst-case running time of the algorithm subquadratic and deterministic. The algorithm does not have to know the value of $\xi$ in advance. Instead, it runs an exponential search on $\xi$, which approximates well the correct value of $\xi$, up to a constant factor. However, this approximation concerns only the size $h$ of the subset $H$ computed in the first stage of the algorithm, whereas the number of the remaining triangles whose union covers $\bigcup T \setminus \bigcup H$ may be much larger than $h$. An open problem that this paper raises is to compute (in subquadratic time) a subset $H' \subset T$ such that $\bigcup H' = \bigcup T$ and $|H'|$ is within a constant (or even $O(\log n)$) factor off the optimum size $\xi$, or, alternatively, to show that this problem is *3SUM-hard*.

In addition, the subset $H$ of triangles that the algorithm computes is not a hitting set for the weighted system $(T, V^*)$ but is rather a $(1/2\xi)$-net for that system. Thus another question that arises is whether the Brönnimann–Goodrich algorithm can be transformed to an algorithm that finds a small $\varepsilon$-net in a general setting (with finite VC-dimension), as it is believed that finding the smallest $\varepsilon$-net is NP-complete. This might lead to an approximation algorithm for finding minimum-size $\varepsilon$-nets.

We showed that our approach can be easily extended to simply shaped bodies of constant description complexity in $\mathbb{R}^d$ for $d \geq 2$, where the union is determined by $\xi$ bodies. In the planar case, the running time remains $O(n^{4/3} \log n + n\xi \log^2 n)$. In $d \geq 3$, the union can be constructed in randomized expected time $O(n^{d-1}\xi \log^2 n)$, which is asymptotically smaller than $n^d$ for $\xi = o(\frac{n}{\log^2 n})$. For $d > 3$, we computed the union recursively on $d$ by constructing the union along each object boundary separately. However, this recursion had to stop at $d = 3$. Indeed, for $d = 3$, applying the two-dimensional algorithm on the boundary of each input body yields an overall $O(n^{7/3} \log n + n^2 \xi \log^2 n)$ expected running time, which is worse than the bound that we have obtained when $\xi = o(\frac{n^{1/3}}{\log n})$. (Note, however, that the two-dimensional approach is used in the actual construction of the union, because a global approach, in this case, would yield an inefficient solution, since it involves the vertical decomposition of simply shaped bodies in three dimensions, which may become quadratic in the number of bodies in the worst case [14, 25, 34].)

A direction for further research is to determine whether there exist simpler efficient approaches to the union construction problem studied in this paper. We note that the standard RIC of [32] may fail in the case that we have considered. In fact, the standard bad example for the RIC, consisting of $n$ triangles that form $\Theta(n^2)$ shallow vertices that are all covered by one large triangle (or, more generally, sparsely covered by $\xi = o(n)$ triangles), shows that the RIC may fail to construct the union in an output-sensitive manner.

Another direction for further research is to extend our approach to instances involving unions in three dimensions where the worst-case complexity of the union is only quadratic or near-quadratic (see [4, 7, 33] for known instances of this kind). Our approach runs in *subcubic* time, when $\xi$ is small, but does not improve upon standard, output-insensitive techniques when the union complexity is always near-quadratic. The simplest instance of such a problem would be the following: Given a collection of $n$ balls in $\mathbb{R}^3$ whose union is equal to the union of some $\xi = o(n)$ of the balls, can the union be constructed in subquadratic time?

Finally, we note that in an earlier version of the algorithm [20], we used a different approach, based on a careful implementation of the DC algorithm of [18]. The previous approach is more complicated, yields a somewhat less efficient solution (which is subquadratic for only a smaller range of the values of the parameter $\xi$), and is more difficult to extend to other geometric shapes and to higher dimensions (in this previous approach, the implementation was based on the techniques of [2, 3, 5, 11, 19]. Our new approach, based on the technique of Brönnimann and Goodrich, is simpler and more generic, improves our previous result, and extends to other shapes and to higher dimensions.

## Appendix A. The actual model for sampling $R$.

In this appendix we show that Lemma 2.1 continues to hold under the actual model of sampling $R$.

As described in section 3, we draw the elements of $R$ by randomly making $\frac{9c'r\binom{n}{2}}{\kappa}$ independent selections of a vertex out of $V^+$ for some constant $c' \geq 1$, where in each trial, each vertex (or more precisely, each pair of triangle edges from distinct triangles) is chosen with probability $\frac{1}{9\binom{n}{2}}$ (thus the same vertex may be sampled more than once). The probability $p$ that a vertex $v \in V^+$ is chosen (at least once) is equal to

$$p = 1 - \left(1 - \frac{1}{9\binom{n}{2}}\right)^{9c'r\frac{\binom{n}{2}}{\kappa}}. \tag{A.1}$$

It is easily checked that $p$ is smaller than $c' \frac{r}{\kappa}$. Moreover, one can also easily show that

$$p > c' \frac{r}{\kappa} - \frac{(c'r)^2}{\kappa^2}. \tag{A.2}$$

In this model, the variables $X_1, \ldots, X_{\kappa_S}$ (as defined in Lemma 2.1) are no longer independent. Nevertheless, examining the proof of the deviation bound given in [6, Theorem A.13], we note that the only place where it uses the assumption that these variables are independent is in the derivation of the equality

$$\mathbf{E}\left[e^{\sum_{i=1}^{\kappa_S} \lambda X_i}\right] = \prod_{i=1}^{\kappa_S} \mathbf{E}\left[e^{\lambda X_i}\right]$$

for any $\lambda$. Moreover, the analysis in [6] uses only the value $\lambda = \frac{r_0}{p\kappa_S}$, where $r_0$ is defined as in Lemma 2.1. An inspection of the derivation of these bounds in [6] shows that they continue to hold when

$$\mathbf{E}\left[e^{\sum_{i=1}^{\kappa_S} \lambda X_i}\right] \le \prod_{i=1}^{\kappa_S} \mathbf{E}\left[e^{\lambda X_i}\right].$$

Furthermore, Lemma 2.1 continues to hold when the weaker inequality

$$\mathbf{E}\left[e^{\sum_{i=1}^{\kappa_S} \lambda X_i}\right] \le \gamma \prod_{i=1}^{\kappa_S} \mathbf{E}\left[e^{\lambda X_i}\right] \tag{A.3}$$

holds for some positive constant $\gamma$. This has the effect of multiplying the probability that (2.1) fails by $\gamma$, which implies that (2.1) still holds, w.o.p. Hence, it suffices to show that (A.3) holds for the above value of $\lambda$. More precisely, as in the original proof of Lemma 2.1, it suffices to establish this under the assumption that $\kappa_S > \frac{\kappa}{t}$.

In our model,

$$\prod_{i=1}^{\kappa_S} \mathbf{E}\left[e^{\lambda X_i}\right] = \left(e^{\lambda} p + (1-p)\right)^{\kappa_S} = \left(1 + p(e^{\lambda} - 1)\right)^{\kappa_S}, \tag{A.4}$$

and

$$\mathbf{E}\left[e^{\sum_{i=1}^{\kappa_S} \lambda X_i}\right] = \sum_{m=0}^{r^*} Pr\left[r_S = m\right] e^{\lambda m}, \tag{A.5}$$

where $r^* = \min\{\frac{9c'r\binom{n}{2}}{\kappa}, \kappa_S\}$. (Note that $Pr[r_S = m] = 0$ for any $m > r^*$.)

In each of the $\frac{9c'r\binom{n}{2}}{\kappa}$ drawing trials, the probability that we have selected a vertex $v$, and that it is not covered by $S$, is $q = \frac{\kappa}{9\binom{n}{2}} \cdot \frac{\kappa_S}{\kappa} = \frac{\kappa_S}{9\binom{n}{2}}$. Since these trials are independent, we have

$$Pr\left[r_S = m\right] = \binom{r^*}{m} q^m (1-q)^{r^*-m}.$$

Hence the expression in (A.5) becomes

$$\sum_{m=0}^{r^*} \binom{r^*}{m} q^m (1-q)^{r^*-m} e^{\lambda m} = \left(e^\lambda q + 1 - q\right)^{r^*} = \left(1 + q\left(e^\lambda - 1\right)\right)^{r^*}.$$

In other words, putting $e^\lambda - 1 = \lambda_0$, we need to show that

$$\left(1+\lambda_0 q\right)^{r^*} \leq \gamma\left(1+\lambda_0 p\right)^{\kappa_S}$$

for some constant $\gamma > 0$. We will show that

$$\left(1+\lambda_0 q\right)^{r^*} \leq \left(1+\lambda_0 p\right)^{2c'r}\left(1+\lambda_0 p\right)^{\kappa_S},$$

which implies the preceding inequality because $(1+\lambda_0 p)^{2c'r} = O(1)$. Indeed, $(1+\lambda_0 p)^{2c'r} < e^{2c'\lambda_0 pr}$. Using the fact that $e^\lambda \leq 1+2\lambda$ for $0 \leq \lambda \leq 1$ and substituting $\lambda = \frac{r_0}{p\kappa_S}$ (which is indeed $\leq 1$ when $r_0$ is chosen as in Lemma 2.1 and $c'$ is sufficiently large, as is easily verified) and $\lambda_0 = e^\lambda - 1$, we have $\lambda_0 \leq 2\lambda$, and thus

$$e^{2c'\lambda_0 pr} \leq e^{4c'\frac{rr_0}{\kappa_S}}.$$

Since we choose $r_0 = \sqrt{2c_0\frac{r}{\kappa}\kappa_S \log n}$ in Lemma 2.1, for some constant $c_0 \geq 1$, the latter expression is smaller than

$$e^{4c'\frac{r}{\kappa_S}\sqrt{2c_0\frac{r}{\kappa}\kappa_S\log n}} = e^{O\left(r\sqrt{\frac{r\log n}{\kappa\kappa_S}}\right)},$$

which, since we assume that $\kappa_S > \frac{\kappa}{t}$, is upper bounded by

$$e^{O\left(\frac{r}{\kappa}\sqrt{tr\log n}\right)}.$$

Substituting $r = ct\log n$, for some constant $c$, and $t = \max\{\frac{\sqrt{\kappa}}{\xi\log n}, 1\}$, as above, and using the assumption that $\kappa > \beta\max\{\xi^2, n^{4/3}\}$, for an absolute constant $\beta > 1$ (see section 3), we have

$$e^{2c'\lambda_0 pr} < \max\left\{e^{O\left(\frac{1}{\xi^2}\right)}, e^{O\left(\frac{\log^2 n}{\kappa}\right)}\right\} = O(1).$$

It thus remains to show that

$$\left(1+\lambda_0 q\right)^{r^*} \leq \left(1+\lambda_0 p\right)^{2c'r+\kappa_S}. \tag{A.6}$$

Let us first assume that $\frac{9c'r\binom{n}{2}}{\kappa} \leq \kappa_S$. We thus need to show that

$$\left(1+\lambda_0\frac{\kappa_S}{9\binom{n}{2}}\right)^{\frac{9c'r\binom{n}{2}}{\kappa}} \leq \left(1+\lambda_0 p\right)^{2c'r+\kappa_S}$$

or that

$$\left(1+\lambda_0\frac{\kappa_S}{9\binom{n}{2}}\right)^{\frac{9\binom{n}{2}}{\kappa_S}} \leq \left(1+\lambda_0 p\right)^{\frac{\kappa(2c'r+\kappa_S)}{c'r\kappa_S}}.$$

Note that the function $(1+\lambda_0 x)^{\frac{1}{x}}$ is monotone decreasing, and since we have assumed that $\frac{9c'r\binom{n}{2}}{\kappa} \le \kappa_S$, it follows that $\frac{\kappa_S}{9\binom{n}{2}} \ge \frac{c'r}{\kappa} > p$. We thus have

$$\left(1+\lambda_0\frac{\kappa_S}{9\binom{n}{2}}\right)^{\frac{9\binom{n}{2}}{\kappa_S}} \le (1+\lambda_0 p)^{\frac{1}{p}}.$$

It therefore suffices to show that $\frac{1}{p} \le \frac{\kappa(2c'r+\kappa_S)}{c'r\kappa_S}$. Using (A.2), $p > c'\frac{r}{\kappa} - \frac{(c'r)^2}{\kappa^2}$, and thus it suffices to show that $\frac{c'r}{\kappa}(1-\frac{c'r}{\kappa}) \ge \frac{c'r\kappa_S}{\kappa(2c'r+\kappa_S)}$, or that

$$1-\frac{c'r}{\kappa} \ge 1-\frac{2c'r}{2c'r+\kappa_S}, \tag{A.7}$$

or that

$$\kappa \ge c'r+\frac{\kappa_S}{2},$$

which clearly holds, since $\kappa_S \le \kappa$ and $r = o(\kappa)$.

We next assume that $\frac{9c'r\binom{n}{2}}{\kappa} > \kappa_S$. We thus need to show that

$$\left(1+\lambda_0\frac{\kappa_S}{9\binom{n}{2}}\right) \le (1+\lambda_0 p)^{\frac{2c'r+\kappa_S}{\kappa_S}}.$$

Using the facts that $(1+\lambda_0 p)^{\frac{2c'r+\kappa_S}{\kappa_S}} \ge 1+\lambda_0 p\,(\frac{2c'r+\kappa_S}{\kappa_S})$ and $\frac{\kappa_S}{9\binom{n}{2}} < \frac{c'r}{\kappa}$, as well as (A.2), it is sufficient to show that

$$\frac{c'r}{\kappa} \le c'\frac{r}{\kappa}\left(1-\frac{c'r}{\kappa}\right)\left(1+\frac{2c'r}{\kappa_S}\right)$$

or that $(1-\frac{c'r}{\kappa})(1+\frac{2c'r}{\kappa_S}) \ge 1$, which is identical to (A.7), as is easily checked, and thus follows by the preceding argument.

We note that (A.6) holds for any value of $\lambda_0 > 0$, and the assumption on $\lambda$ is used only when showing that $(1+\lambda_0 p)^{2c'r} = O(1)$. This completes the proof of (A.3) for the value of $\lambda$ that we use and therefore shows that Lemma 2.1 also holds for the sampling model used by our algorithm.

**Acknowledgments.** The authors wish to thank Ken Clarkson, Alon Efrat, and Sariel Har-Peled for useful discussions on this problem. In particular, Sariel's insistence that we use the Brönnimann–Goodrich technique (instead of the DC algorithm in the earlier version [20]) has finally led to the improved algorithm presented in this paper. The authors also wish to thank two anonymous referees for useful comments.

## REFERENCES

[1] P. K. Agarwal and S. Har-Peled, *Two Randomized Incremental Algorithms for Planar Arrangements, with a Twist*, manuscript, 2001.
[2] P. K. Agarwal and J. Matoušek, *On range searching with semialgebraic sets*, Discrete Comput. Geom., 11 (1994), pp. 393–418.
[3] P. K. Agarwal, M. Pellegrini, and M. Sharir, *Counting circular arc intersections*, SIAM J. Comput., 22 (1993), pp. 778–793.

[4] P. K. Agarwal and M. Sharir, *Pipes, cigars, and Kreplach: The union of Minkowski sums in three dimensions*, Discrete Comput. Geom., 24 (2000), pp. 645–685.

[5] P. K. Agarwal, M. Sharir, and S. Toledo, *Applications of parametric searching in geometric optimization*, J. Algorithms, 17 (1994), pp. 292–318.

[6] N. Alon and J. H. Spencer, *The Probabilistic Method*, 2nd ed., Wiley-Interscience, New York, 2000.

[7] B. Aronov, A. Efrat, V. Koltun, and M. Sharir, *On the union of $\kappa$-round objects in three and four dimensions*, Discrete Comput. Geom., to appear.

[8] P. Assouad, *Density and dimension*, Ann. Inst. Fourier (Grenoble), 33 (1983), pp. 233–282.

[9] J. Bentley and T. Ottmann, *Algorithms for reporting and counting geometric intersections*, IEEE Trans. Comput., 28 (1979), pp. 643–647.

[10] H. Brönnimann and M. Goodrich, *Almost optimal set covers in finite VC-dimension*, Discrete Comput. Geom., 14 (1995), pp. 463–479.

[11] B. Chazelle and J. Friedman, *A deterministic view of random sampling and its use in geometry*, Combinatorica, 10 (1990), pp. 229–249.

[12] K. L. Clarkson, *Algorithms for polytope covering and approximation*, in Proceedings of the 3rd Workshop on Algorithms and Data Structures, Lecture Notes in Comput. Sci. 709, Springer-Verlag, Berlin, 1993, pp. 246–252.

[13] K. L. Clarkson, *Las Vegas algorithms for linear and integer programming when the dimension is small*, J. ACM, 42 (1995), pp. 488–499.

[14] M. de Berg, L. J. Guibas, and D. Halperin, *Vertical decompositions for triangles in* 3*-space*, Discrete Comput. Geom., 15 (1996), pp. 35–61.

[15] M. de Berg, M. Katz, A. F. van der Stappen, and J. Vleugels, *Realistic input models for geometric algorithms*, Algorithmica, 34 (2002), pp. 81–97.

[16] M. de Berg, M. van Kreveld, M. Overmars, and O. Schwarzkopf, *Computational Geometry: Algorithms and Applications*, 2nd ed., Springer-Verlag, Berlin, 2000.

[17] G. Even, D. Rawitz, and M. Shahar, *Hitting sets when the VC-dimension is small*, Inform. Process. Lett., to appear.

[18] E. Ezra, D. Halperin, and M. Sharir, *Speeding up the incremental construction of the union of geometric objects in practice*, Comput. Geom., 27 (2004), pp. 63–85.

[19] E. Ezra and M. Sharir, *Counting and representing intersections among triangles in three dimensions*, Comput. Geom., to appear.

[20] E. Ezra and M. Sharir, *Output-sensitive construction of the union of triangles*, in Proceedings of the 15th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'04), ACM, New York, SIAM, Philadelphia, 2004, pp. 413–422.

[21] A. Gajentaan and M. H. Overmars, *On a class of $O(n^2)$ problems in computational geometry*, Comput. Geom., 5 (1995), pp. 165–185.

[22] D. Haussler and E. Welzl, *$\varepsilon$-nets and simplex range queries*, Discrete Comput. Geom., 2 (1987), pp. 127–151.

[23] D. P. Huttenlocher, K. Kedem, and M. Sharir, *The upper envelope of Voronoi surfaces and its applications*, Discrete Comput. Geom., 9 (1993), pp. 267–291.

[24] K. Kedem, R. Livne, J. Pach, and M. Sharir, *On the union of Jordan regions and collision-free translational motion amidst polygonal obstacles*, Discrete Comput. Geom., 1 (1986), pp. 59–71.

[25] V. Koltun, *Almost tight upper bounds for vertical decompositions in four dimensions*, J. ACM, 51 (2004), pp. 699–730.

[26] J.-C. Latombe, *Robot Motion Planning*, Kluwer Academic Publishers, Boston, 1991.

[27] N. Littlestone, *Learning quickly when irrelevant attributes abound: A new linear-threshold algorithm*, Mach. Learn., 2 (1988), pp. 285–318.

[28] J. Matoušek, N. Miller, J. Pach, M. Sharir, S. Sifrony, and E. Welzl, *Fat triangles determine linearly many holes*, in Proceedings of the 32nd Annual IEEE Symposium on Foundations in Computer Science, IEEE Comput. Soc. Press, Los Alamitos, CA, 1991, pp. 49–58.

[29] J. Matoušek, J. Pach, M. Sharir, S. Sifrony, and E. Welzl, *Fat triangles determine linearly many holes*, SIAM J. Comput., 23 (1994), pp. 154–169.

[30] J. Matoušek, *Cutting hyperplane arrangements*, Discrete Comput. Geom., 6 (1991), pp. 385–406.

[31] J. Matoušek, *Lectures on Discrete Geometry*, Grad. Texts in Math. 212, Springer-Verlag, New York, 2002.

[32] K. Mulmuley, *Computational Geometry: An Introduction Through Randomized Algorithms*, Prentice–Hall, Englewood Cliffs, NJ, 1994.

[33] J. Pach, I. Safruti, and M. Sharir, *The union of congruent cubes in three dimensions*, Discrete Comput. Geom., 30 (2003), pp. 133–160.

[34] B. Tagansky, *A new technique for analyzing substructures in arrangements of piecewise linear surfaces*, Discrete Comput. Geom., 16 (1996), pp. 455–479.

[35] E. Welzl, *Partition trees for triangle counting and other range searching problems*, in Proceedings of the 4th Annual ACM Symposium on Computational Geometry, ACM, New York, 1988, pp. 23–33.

SIAM J. COMPUT. © 2005 Society for Industrial and Applied Mathematics
Vol. 34, No. 6, pp. 1352–1369

# EXTENDING DOWNWARD COLLAPSE FROM 1-VERSUS-2 QUERIES TO $m$-VERSUS-$m$+1 QUERIES*

EDITH HEMASPAANDRA†, LANE A. HEMASPAANDRA‡, AND HARALD HEMPEL§

**Abstract.** The top part of Figure 1.1 shows some classes from the (truth-table) bounded-query and boolean hierarchies. It is well known that if either of these hierarchies collapses at a given level, then all higher levels of that hierarchy collapse to that same level. This is a standard "upward translation of equality" that has been known for over a decade. The issue of whether these hierarchies can translate equality *downwards* has proven vastly more challenging. In particular, with regard to Figure 1.1, consider the following claim:

$$\mathrm{P}^{\Sigma_k^p}_{m\text{-tt}} = \mathrm{P}^{\Sigma_k^p}_{m+1\text{-tt}} \Rightarrow \mathrm{DIFF}_m(\Sigma_k^p) = \mathrm{coDIFF}_m(\Sigma_k^p) = \mathrm{BH}(\Sigma_k^p). \qquad (*)$$

This claim, if true, says that equality translates downwards between levels of the bounded-query hierarchy and the boolean hierarchy levels that (before the fact) are immediately below them.

Until recently, it was not known whether (*) *ever* held, except for the degenerate cases $m = 0$ and $k = 0$. Then Hemaspaandra, Hemaspaandra, and Hempel [*SIAM J. Comput.,* 28 (1999), pp. 383–393] proved that (*) holds for all $m$, for $k > 2$. Buhrman and Fortnow [*J. Comput. System Sci.,* 59 (1999), pp. 182–199] then showed that, when $k = 2$, $(*)$ holds for the case $m = 1$. In this paper, we prove that for the case $k = 2$, $(*)$ holds for all values of $m$. Since there is an oracle relative to which "for $k = 1$, $(*)$ holds for all $m$"fails (see Buhrman and Fortnow), our achievement of the $k = 2$ case cannot be strengthened to $k = 1$ by any relativizable proof technique. The new downward translation we obtain also tightens the collapse in the polynomial hierarchy implied by a collapse in the bounded-query hierarchy of the second level of the polynomial hierarchy.

**Key words.** computational complexity theory, no-search easy-hard technique, downward collapse, upward separation, downward translation of equality, boolean hierarchy, polynomial hierarchy

**AMS subject classifications.** 68Q15, 68Q10, 03D15

**DOI.** 10.1137/S0097539701391002

## 1. Introduction.

Does the equality of low-complexity classes imply the equality of higher-complexity classes? Does the equality of high-complexity classes imply the equality of lower-complexity classes? These questions—known, respectively, as upward and downward translation of equality—have long been central topics in computational complexity theory. For example, in the seminal paper on the polynomial

*Received by the editors June 15, 2001; accepted for publication (in revised form) September 30, 2004; published electronically July 26, 2005. Earlier versions of parts of this paper have appeared in the Sixth Italian Conference on Theoretical Computer Science (ICTCS '98) and the Sixteenth Annual Symposium on Theoretical Aspects of Computer Science (STACS '99). This work was supported in part by grants NSF-CCR-9322513, DAAD-315-PRO-fo-ab/NSF-INT-9513368, DAAD-315-PPP-gü-ab/NSF-INT-9815095, NSF-CCR-0311021, and NSF-CCF-0426761 and by a NATO Postdoctoral Science Fellowship from the "Gemeinsames Hochschulsonderprogramm III von Bund und Ländern" program of the DAAD.

http://www.siam.org/journals/sicomp/34-6/39100.html

†Department of Computer Science, Rochester Institute of Technology, Rochester, NY 14623 (eh@cs.rit.edu). This author's work was done in part while visiting Friedrich-Schiller-Universität and Julius-Maximilians-Universität.

‡Department of Computer Science, University of Rochester, Rochester, NY 14627 (lane@cs.rochester.edu). This author's work was done in part while visiting Friedrich-Schiller-Universität and Julius-Maximilians-Universität.

§Institut für Informatik, Friedrich-Schiller-Universität Jena, 07743 Jena, Germany (hempel@informatik.uni-jena.de). This author's work was done in part while visiting Le Moyne College, and while visiting the University of Rochester under a NATO Postdoctoral Science Fellowship from the DAAD.

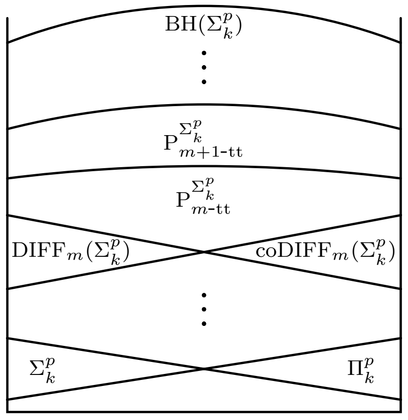

$\mathrm{BH}(\Sigma_k^p)$
$\mathrm{P}^{\Sigma_k^p}_{m+1\text{-tt}}$
$\mathrm{P}^{\Sigma_k^p}_{m\text{-tt}}$
$\mathrm{DIFF}_m(\Sigma_k^p)$
$\mathrm{coDIFF}_m(\Sigma_k^p)$
$\Sigma_k^p$
$\Pi_k^p$

FIG. 1.1. *The figure referred to in the abstract.*

hierarchy, Meyer and Stockmeyer [20] proved that the polynomial hierarchy displays upward translation of equality (e.g., $\mathrm{P} = \mathrm{NP} \Rightarrow \mathrm{P} = \mathrm{PH}$).

The issue of whether the polynomial hierarchy—its levels and/or bounded access to its levels—ever displays *downward* translation of equality has proven more difficult. The first such result regarding bounded access to polynomial hierarchy levels was recently obtained by Hemaspaandra, Hemaspaandra, and Hempel [14], who proved that if for some high level of the polynomial hierarchy one query equals two queries, then the hierarchy collapses down not just to one query to that level, but rather to that level itself. That is, they proved the following result (note: the levels of the polynomial hierarchy [20, 27] are denoted in the standard way, namely, $\Sigma_0^p = \mathrm{P}$, $\Sigma_1^p = \mathrm{NP}$, $\Sigma_k^p = \mathrm{NP}^{\Sigma_{k-1}^p}$ for each $k > 1$, and $\Pi_k^p = \{L \mid \overline{L} \in \Sigma_k^p\}$ for each $k \geq 0$).

THEOREM 1.1 (see [14]). *For each $k > 2$: If $\mathrm{P}^{\Sigma_k^p[1]} = \mathrm{P}^{\Sigma_k^p[2]}$, then $\Sigma_k^p = \Pi_k^p = \mathrm{PH}$.*

There are two clear directions in which one might hope to strengthen Theorem 1.1. First, one might ask not just about 1-versus-2 queries but rather about $m$-versus-$m+1$ queries. Second, one might ask if the $k > 2$ can be improved to $k > 1$. Both of these have been achieved. The first strengthening was achieved in a more technical section of the same paper by Hemaspaandra, Hemaspaandra, and Hempel [14]. They showed that Theorem 1.1 was just a special case of a more general downward translation result they established, for $k > 2$, between bounded access to $\Sigma_k^p$ and the boolean hierarchy over $\Sigma_k^p$. The second type of strengthening was achieved by Buhrman and Fortnow [4], who showed that Theorem 1.1 holds even for $k = 2$, but who also showed that no relativizable technique can establish Theorem 1.1 for $k = 1$.

Neither of the results or proofs just mentioned is broad enough to achieve both strengthenings simultaneously. In this paper we derive new theorems strong enough to achieve this. In particular, we unify and extend all the above claims, and from our more general results it easily follows that (see Corollary 5.1)

For each $m > 0$ and each $k > 1$;

$$\mathrm{P}^{\Sigma_k^p}_{m\text{-tt}} = \mathrm{P}^{\Sigma_k^p}_{m+1\text{-tt}} \Rightarrow \mathrm{DIFF}_m(\Sigma_k^p) = \mathrm{coDIFF}_m(\Sigma_k^p).$$

Prior to the work of this paper, the cases $(k = 2 \wedge m = 2)$, $(k = 2 \wedge m = 3)$, $(k = 2 \wedge m = 4)$, and so on were open. As shown near the end of section 5, the stronger downward translation we obtain yields a strengthened collapse of the polynomial hierarchy under the assumption of a collapse in the bounded-query hierarchy over $\mathrm{NP}^{\mathrm{NP}}$. (Throughout this paper, we mean the *truth-table* bounded-query hierarchy when we say bounded-query hierarchy. However, as mentioned in section 5, our results equally well strengthen the collapse of the polynomial hierarchy under the assumption of a collapse in the *Turing* bounded-query hierarchy over $\mathrm{NP}^{\mathrm{NP}}$.)

The results that lead to the result mentioned above (i.e., Corollary 5.1) are themselves examples of downward translation of equality. Those intermediate results that are of interest in their own right are proven in sections 3 and 4.

We conclude this section with some comments and pointers to the literature. As to techniques, to study *upward* translations of equality resulting from collapses of the boolean hierarchy, Kadin [17] introduced what is known as the "easy-hard technique," and that technique was employed and strengthened in a long series of papers by many authors (see the survey [13]). In particular, Hemaspaandra, Hemaspaandra, and Hempel [14] achieved Theorem 1.1 by introducing what can be called the *no-search easy-hard technique*—basically, they show that a complexity-raising search seemingly central in the easy-hard technique can in fact be eliminated, and this stronger technique opens the door to stronger results, namely, *downward* translation of equality results for bounded-query classes. The present paper builds on the no-search easy-hard technique and on a variation on that used by Buhrman and Fortnow to prove the 1-versus-2-queries case at the second level of the polynomial hierarchy. However, these approaches seem not to be strong enough to yield our result, and so we also must combine with these techniques a new approach extending beyond 1-versus-2 queries.[1] We mention that Chang has obtained exciting applications of easy-hard-type arguments in the context of the study of approximation [7]. We also mention that there is a body of literature showing that equality of exponential-time classes translates downwards in a limited sense: Relationships are obtained connecting such equalities to whether *sparse* sets within lower time classes are unexpectedly simple (see [9, 10], see also [6, 21, 22]; limitations of this line are presented in [1, 2, 16]). Other than being an interesting restricted type of downward translation of equality, that work has no close connection with the present paper due to that body of literature being applicable only to sparse sets. Finally, we mention that the study of downward collapse results is closely related to the formal study of the power of query order—whether the order in which databases are accessed matters—an area recently introduced by Hemaspaandra, Hempel, and Wechsung [15]. In particular, downward collapse techniques have been used to understand the power of query order within the polynomial hierarchy (see [12], the survey [11], and the references therein, especially [31]).

## 2. Preliminaries.

To explain exactly what we do and how it extends previous results, we now state the previous results in the more general forms in which they were

[1] Regarding this new approach (and this footnote is aimed primarily at those already familiar with the techniques of the previous papers on the no-search easy-hard technique): In the previous work extending Theorem 1.1 to the boolean hierarchy (part 1 of Theorem 2.4), the "coordination" difficulties presented by the fact that boolean hierarchy sets are in effect handled via collections of machines were resolved via using certain lexicographically extreme objects as clear signposts to signal machines with (see [14, section 3]). In the current stronger context that approach fails. Instead, we integrate into the structure of no-search easy-hard-technique proofs (especially those of [14, 4]) the so-called "telescoping" normal form possessed by the boolean hierarchy over $\Sigma_k^p$ (for each $k$, see [18, 5]). (The telescoping normal form guarantees that if $L \in \mathrm{DIFF}_m(\Sigma_k^p)$, then there are sets $L_1, L_2, \ldots, L_m \in \Sigma_k^p$ such that $L = L_1 - (L_2 - (L_3 - \cdots (L_{m-1} - L_m)\cdots))$ and $L_1 \supseteq L_2 \supseteq \cdots \supseteq L_{m-1} \supseteq L_m$.) This normal form has in different contexts proven useful in the study of boolean hierarchies (see, e.g., [5, 6, 18]) and has been used by Rohatgi in the context of a paper using the original (i.e., the with-search version of the) easy-hard technique [24].

actually established, though in some cases with different notation or statements (see, e.g., the interesting paper by Wagner [31] regarding the relationship between truth-table classes and what in the present paper is $\bigoplus$ notation). Before stating the results, we very briefly remind the reader of some definitions and notations, namely the $\Delta$ levels of the polynomial hierarchy, truth-table access, symmetric difference classes, and boolean hierarchies. A detailed introduction to the boolean hierarchy, including its motivation and applications, can be found in [5, 6].

DEFINITION 2.1.

1. *For each* $k \geq 1$, $\Delta_k^p$ *denotes* $\mathrm{P}^{\Sigma_{k-1}^p}$ [20]. *For each* $m \geq 0$ *and each set* $A$, $\mathrm{P}_{m\text{-tt}}^A$ *denotes the class of languages accepted by deterministic polynomial-time machines allowed* $m$ *truth-table (i.e., nonadaptive) queries to* $A$ *(see* [19]*). For each* $m \geq 0$ *and each complexity class* $\mathcal{C}$, $\mathrm{P}_{m\text{-tt}}^{\mathcal{C}}$ *is defined as* $\bigcup_{A \in \mathcal{C}} \mathrm{P}_{m\text{-tt}}^A$. *For each* $m \geq 0$ *and each set* $A$, $\mathrm{P}^{A[m]}$ *denotes the class of languages accepted by deterministic polynomial-time machines allowed* $m$ *Turing (i.e., adaptive) queries to* $A$ *(see* [19]*). For each* $m \geq 0$ *and each complexity class* $\mathcal{C}$, $\mathrm{P}^{\mathcal{C}[m]}$ *is defined as* $\bigcup_{A \in \mathcal{C}} \mathrm{P}^{A[m]}$.
2. *For any sets* $C$ *and* $D$, $C \oplus D = (C - D) \cup (D - C)$. *For any classes* $\mathcal{C}$ *and* $\mathcal{D}$, $\mathcal{C} \bigoplus \mathcal{D} = \{L \mid (\exists C \in \mathcal{C})(\exists D \in \mathcal{D})[L = C \oplus D]\}$. *We will refer to classes defined via* $\bigoplus$ *as* symmetric difference classes.
3. [5, 6] *Let* $\mathcal{C}$ *be any complexity class. The levels of the boolean hierarchy are defined as follows.*
   (a) $\mathrm{DIFF}_1(\mathcal{C}) = \mathcal{C}$.
   (b) *For all* $m \geq 1$, $\mathrm{DIFF}_{m+1}(\mathcal{C}) = \{L \mid (\exists L_1 \in \mathcal{C})(\exists L_2 \in \mathrm{DIFF}_m(\mathcal{C}))[L = L_1 - L_2]\}$.
   (c) *For all* $m \geq 1$, $\mathrm{coDIFF}_m(\mathcal{C}) = \{L \mid \overline{L} \in \mathrm{DIFF}_m(\mathcal{C})\}$.
   (d) $\mathrm{BH}(\mathcal{C})$, *the boolean hierarchy over* $\mathcal{C}$, *is* $\bigcup_{m \geq 1} \mathrm{DIFF}_m(\mathcal{C})$.

As is standard, for any set $A$ and any natural number $n$, $A^{\leq n}$ denotes the set of all strings in $A$ of length less than or equal to $n$. As is standard, for any set $A$ and any natural number $n$, $A^{=n}$ denotes the set of all strings in $A$ of length exactly $n$. For example, note that "$|x| = n$," "$x \in (\Sigma^*)^{=n}$," and "$x \in \Sigma^n$" are all ways of saying that the length of string $x$ equals $n$. This notation should not be confused with the notation $\Sigma_n^p$, which denotes the $n$th level of the polynomial hierarchy [20].

The relationship between the levels of the boolean hierarchy over $\Sigma_k^p$, bounded access to $\Sigma_k^p$, and various symmetric difference classes is as follows.

PROPOSITION 2.2.

1. [30] *For each* $k \geq 1$ *and each* $m \geq 1$, $\mathrm{P}_{m\text{-tt}}^{\Sigma_k^p} \subseteq \mathrm{DIFF}_{m+1}(\Sigma_k^p) \subseteq \mathrm{P}_{m+1\text{-tt}}^{\Sigma_k^p}$ *and* $\mathrm{P}_{m\text{-tt}}^{\Sigma_k^p} \subseteq \mathrm{coDIFF}_{m+1}(\Sigma_k^p) \subseteq \mathrm{P}_{m+1\text{-tt}}^{\Sigma_k^p}$.
2. [18] *For all* $k \geq 1$ *and all* $m \geq 1$,
$$\mathrm{DIFF}_m(\Sigma_k^p) = \underbrace{\Sigma_k^p \bigoplus \Sigma_k^p \bigoplus \cdots \bigoplus \Sigma_k^p}_{m \text{ times}}.$$
3. [31] *For all* $k \geq 1$ *and all* $m \geq 1$,
$$\mathrm{P}_{m\text{-tt}}^{\Sigma_k^p} = \mathrm{P} \bigoplus \mathrm{DIFF}_m(\Sigma_k^p) = \mathrm{P} \bigoplus \underbrace{\Sigma_k^p \bigoplus \Sigma_k^p \bigoplus \cdots \bigoplus \Sigma_k^p}_{m \text{ times}}.$$

We will use the following easy fact about symmetric difference classes.

*Observation* 2.3. Let $\mathcal{C}_1$, $\mathcal{C}_2$, and $\mathcal{D}$ be complexity classes. If $\mathcal{C}_1 \subseteq \mathcal{C}_2$, then $\mathcal{C}_1 \bigoplus \mathcal{D} \subseteq \mathcal{C}_2 \bigoplus \mathcal{D}$.

Now we can state what the earlier papers achieved. (In achieving these results, those papers obtained as corollaries the results attributed to them in the Introduction.)

THEOREM 2.4.

1. [14] *Let $m > 0$, $0 \leq i < j < k$, and $i < k-2$. If* $\mathrm{P}_{1\text{-tt}}^{\Sigma_i^p} \bigoplus \mathrm{DIFF}_m(\Sigma_k^p) = \mathrm{P}_{1\text{-tt}}^{\Sigma_j^p} \bigoplus \mathrm{DIFF}_m(\Sigma_k^p)$, *then* $\mathrm{DIFF}_m(\Sigma_k^p) = \mathrm{coDIFF}_m(\Sigma_k^p)$.
2. [4] *If* $\mathrm{P} \bigoplus \Sigma_2^p = \mathrm{NP} \bigoplus \Sigma_2^p$, *then* $\Sigma_2^p = \Pi_2^p = \mathrm{PH}$.

In this paper, we unify both of the above results—and achieve the strengthened corollary alluded to in the Introduction regarding the relative power of $m$ and $m+1$ queries to $\Sigma_k^p$ (namely Corollary 5.1: For each $m > 0$ and each $k > 1$, $\mathrm{P}_{m\text{-tt}}^{\Sigma_k^p} = \mathrm{P}_{m+1\text{-tt}}^{\Sigma_k^p} \Rightarrow \mathrm{DIFF}_m(\Sigma_k^p) = \mathrm{coDIFF}_m(\Sigma_k^p)$)—by proving the following downward translation of equality result, which will be our Theorem 3.3:

Let $m > 0$ and $0 < i < k$. If $\Delta_i^p \bigoplus \mathrm{DIFF}_m(\Sigma_k^p) = \Sigma_i^p \bigoplus \mathrm{DIFF}_m(\Sigma_k^p)$, then $\mathrm{DIFF}_m(\Sigma_k^p) = \mathrm{coDIFF}_m(\Sigma_k^p)$.

## 3. A new downward translation of equality.

We first need a definition and a useful lemma.

DEFINITION 3.1. *For any sets $C$ and $D$,* $C \tilde{\oplus} D = \{\langle x, y\rangle \mid x \in C \Leftrightarrow y \notin D\}$.

LEMMA 3.2. *If $C$ is $\leq_{\mathrm{m}}^p$-complete for $\mathcal{C}$ and $D$ is $\leq_{\mathrm{m}}^p$-complete for $\mathcal{D}$, then $C \tilde{\oplus} D$ is $\leq_{\mathrm{m}}^p$-hard for $\mathcal{C} \bigoplus \mathcal{D}$.*

*Proof.* Let $L \in \mathcal{C} \bigoplus \mathcal{D}$. We need to show that $L \leq_{\mathrm{m}}^p C \tilde{\oplus} D$. Let $\widehat{C} \in \mathcal{C}$ and $\widehat{D} \in \mathcal{D}$ be such that $L = \widehat{C} \oplus \widehat{D}$. Let $\widehat{C} \leq_{\mathrm{m}}^p C$ by $f_C$, and $\widehat{D} \leq_{\mathrm{m}}^p D$ by $f_D$. Then $x \in L$ if and only if $x \in \widehat{C} \oplus \widehat{D}$, and $x \in \widehat{C} \oplus \widehat{D}$ if and only if $(x \in \widehat{C} \Leftrightarrow x \notin \widehat{D})$, and $(x \in \widehat{C} \Leftrightarrow x \notin \widehat{D})$ if and only if $(f_C(x) \in C \Leftrightarrow f_D(x) \notin D)$, and $(f_C(x) \in C \Leftrightarrow f_D(x) \notin D)$ if and only if $\langle f_C(x), f_D(x)\rangle \in C \tilde{\oplus} D$. It follows that $x \in L$ if and only if $\langle f_C(x), f_D(x)\rangle \in C \tilde{\oplus} D$. □

We now state our main result. (Note that as both $\Delta_i^p$ and $\Sigma_i^p$ contain both $\emptyset$ and $\Sigma^*$, it is clear that the classes involved in the first equality below are at least as large as the classes involved in the second equality below.)

THEOREM 3.3. *Let $m > 0$ and $0 < i < k$. If* $\Delta_i^p \bigoplus \mathrm{DIFF}_m(\Sigma_k^p) = \Sigma_i^p \bigoplus \mathrm{DIFF}_m(\Sigma_k^p)$, *then* $\mathrm{DIFF}_m(\Sigma_k^p) = \mathrm{coDIFF}_m(\Sigma_k^p)$.

This result almost follows from the forthcoming Theorem 4.1 (which states that if $s, m > 0$, $0 < i < k-1$, and $\mathrm{DIFF}_s(\Sigma_i^p) \bigoplus \mathrm{DIFF}_m(\Sigma_k^p)$ is closed under complementation, then $\mathrm{DIFF}_m(\Sigma_k^p) = \mathrm{coDIFF}_m(\Sigma_k^p)$)—or, to be more accurate, most of its cases are easy corollaries of Theorem 4.1. The $s = 1$ case of Theorem 4.1 states that for all $m > 0$ and all $i$ and $k$ such that $0 < i < k-1$, if $\Sigma_i^p \bigoplus \mathrm{DIFF}_m(\Sigma_k^p)$ is closed under complementation, then $\mathrm{DIFF}_m(\Sigma_k^p) = \mathrm{coDIFF}_m(\Sigma_k^p)$. Since $\Delta_i^p \bigoplus \mathcal{C}$ is closed under complementation for all $\mathcal{C}$ and all $i \geq 0$, we have that if $\Delta_i^p \bigoplus \mathrm{DIFF}_m(\Sigma_k^p) = \Sigma_i^p \bigoplus \mathrm{DIFF}_m(\Sigma_k^p)$, then $\Sigma_i^p \bigoplus \mathrm{DIFF}_m(\Sigma_k^p)$ is closed under complementation. Thus, Theorem 4.1 certainly implies Theorem 3.3 for all $m > 0$ and all $i$ and $k$ such that $0 < i < k-1$. It remains for us to establish the missing cases, and Theorem 3.4 below does exactly that.

THEOREM 3.4. *Let $m > 0$ and $k > 1$. If* $\Delta_{k-1}^p \bigoplus \mathrm{DIFF}_m(\Sigma_k^p) = \Sigma_{k-1}^p \bigoplus \mathrm{DIFF}_m(\Sigma_k^p)$, *then* $\mathrm{DIFF}_m(\Sigma_k^p) = \mathrm{coDIFF}_m(\Sigma_k^p)$.

Before proving Theorem 3.4, we fix some complete languages that will be useful, and establish names that we will use globally for these fixed, complete languages. (In light of the standard quantifier characterization of the polynomial hierarchy's levels [32] and the legality of padding sets to get new sets for which linear-bounded quantification suffices, it is not hard to see that there exist complete languages having the properties stated in Fact 3.5.) Throughout the paper we will use the names introduced in this fact for these sets (or more specifically, let us consider sets of the form guaranteed by the fact to be fixed, and let us attach to them the names used in the fact).

FACT 3.5. *For each $k > 1$, there exist a set $L'_{\Sigma_k^p}$ that is $\leq_{\mathrm{m}}^p$-complete for $\Sigma_k^p$, a set $\widehat{L}_{\Sigma_{k-1}^p}$ that is $\leq_{\mathrm{m}}^p$-complete for $\Sigma_{k-1}^p$, and a set $L''_{\Sigma_{k-2}^p}$ that is $\leq_{\mathrm{m}}^p$-complete for $\Sigma_{k-2}^p$, such that these sets satisfy*

$$L'_{\Sigma_k^p} = \{x \mid (\exists y \in \Sigma^{|x|})(\forall z \in \Sigma^{|x|})[\langle x, y, z\rangle \in L''_{\Sigma_{k-2}^p}]\}$$

*and*

$$\widehat{L}_{\Sigma_{k-1}^p} = \{\langle x, y, z\rangle \mid |x| = |y| \wedge (\exists z')[(|x| = |zz'|) \wedge \langle x, y, zz'\rangle \notin L''_{\Sigma_{k-2}^p}]\}.$$

*Proof of Theorem* 3.4. Let $m > 0$ and $k > 1$. Let $\widehat{L}_{\Sigma_{k-1}^p} \in \Sigma_{k-1}^p$ be as fixed in Fact 3.5, and let $L_{\Delta_{k-1}^p}$ and $L_{\mathrm{DIFF}_m(\Sigma_k^p)}$ be any fixed $\leq_{\mathrm{m}}^p$-complete sets for $\Delta_{k-1}^p$ and $\mathrm{DIFF}_m(\Sigma_k^p)$, respectively; such languages exist, e.g., via the standard type of canonical-complete-set constructions involving padding and enumerations of clocked machines. From Lemma 3.2 it follows that $L_{\Delta_{k-1}^p} \tilde{\oplus} L_{\mathrm{DIFF}_m(\Sigma_k^p)}$ is $\leq_{\mathrm{m}}^p$-hard for $\Delta_{k-1}^p \bigoplus \mathrm{DIFF}_m(\Sigma_k^p)$. Since $\widehat{L}_{\Sigma_{k-1}^p} \tilde{\oplus} L_{\mathrm{DIFF}_m(\Sigma_k^p)} \in \Sigma_{k-1}^p \bigoplus \mathrm{DIFF}_m(\Sigma_k^p)$ and by assumption $\Delta_{k-1}^p \bigoplus \mathrm{DIFF}_m(\Sigma_k^p) = \Sigma_{k-1}^p \bigoplus \mathrm{DIFF}_m(\Sigma_k^p)$, there exists a (polynomial-time) many-one reduction $h$ from $\widehat{L}_{\Sigma_{k-1}^p} \tilde{\oplus} L_{\mathrm{DIFF}_m(\Sigma_k^p)}$ to $L_{\Delta_{k-1}^p} \tilde{\oplus} L_{\mathrm{DIFF}_m(\Sigma_k^p)}$ (in light of the latter's $\leq_{\mathrm{m}}^p$-hardness). So, for all $x_1, x_2, y_1, y_2 \in \Sigma^*$: If $h(\langle x_1, x_2\rangle) = \langle y_1, y_2\rangle$, then $((x_1 \in \widehat{L}_{\Sigma_{k-1}^p} \Leftrightarrow x_2 \notin L_{\mathrm{DIFF}_m(\Sigma_k^p)})$ iff $(y_1 \in L_{\Delta_{k-1}^p} \Leftrightarrow y_2 \notin L_{\mathrm{DIFF}_m(\Sigma_k^p)}))$. Equivalently, for all $x_1, x_2, y_1, y_2 \in \Sigma^*$: If $h(\langle x_1, x_2\rangle) = \langle y_1, y_2\rangle$, then

$$(3.1)\quad (x_1 \in \widehat{L}_{\Sigma_{k-1}^p} \Leftrightarrow x_2 \in L_{\mathrm{DIFF}_m(\Sigma_k^p)}) \text{ iff } (y_1 \in L_{\Delta_{k-1}^p} \Leftrightarrow y_2 \in L_{\mathrm{DIFF}_m(\Sigma_k^p)}).$$

We can use $h$ to recognize some of $\overline{L_{\mathrm{DIFF}_m(\Sigma_k^p)}}$ by a $\mathrm{DIFF}_m(\Sigma_k^p)$ algorithm. In particular, we say that a string $x$ is *easy for length* $n$ if there exists a string $x_1$ such that $|x_1| \leq n$ and $(x_1 \in \widehat{L}_{\Sigma_{k-1}^p} \Leftrightarrow y_1 \notin L_{\Delta_{k-1}^p})$, where $h(\langle x_1, x\rangle) = \langle y_1, y_2\rangle$.

Let $p$ be a fixed polynomial, which will be exactly specified later in the proof. We have the following algorithm to test whether $x \in \overline{L_{\mathrm{DIFF}_m(\Sigma_k^p)}}$ in the case that (our input) $x$ is an easy string for length $p(|x|)$. Guess $x_1$ with $|x_1| \leq p(|x|)$, let $h(\langle x_1, x\rangle) = \langle y_1, y_2\rangle$, and accept if and only if $((x_1 \in \widehat{L}_{\Sigma_{k-1}^p} \Leftrightarrow y_1 \notin L_{\Delta_{k-1}^p}) \wedge y_2 \in L_{\mathrm{DIFF}_m(\Sigma_k^p)})$. (To understand what is going on here, simply note that if $(x_1 \in \widehat{L}_{\Sigma_{k-1}^p} \Leftrightarrow y_1 \notin L_{\Delta_{k-1}^p})$ holds, then by (3.1) we have $x \in \overline{L_{\mathrm{DIFF}_m(\Sigma_k^p)}} \Leftrightarrow y_2 \in L_{\mathrm{DIFF}_m(\Sigma_k^p)}$. Note also that both $x_1 \in \widehat{L}_{\Sigma_{k-1}^p}$ and $y_1 \notin L_{\Delta_{k-1}^p}$ can be very easily tested by a machine that has a $\Sigma_{k-1}^p$ oracle.) This algorithm is not necessarily a $\mathrm{DIFF}_m(\Sigma_k^p)$ algorithm, but it does inspire the following $\mathrm{DIFF}_m(\Sigma_k^p)$ algorithm to test whether $x \in \overline{L_{\mathrm{DIFF}_m(\Sigma_k^p)}}$ in the case that $x$ is an easy string for length $p(|x|)$. For clarity, we will attempt to telegraph the complexity issues as we work to develop the algorithm.

Let $L_1, L_2, \ldots, L_m$ be languages in $\Sigma_k^p$ such that $L_{\mathrm{DIFF}_m(\Sigma_k^p)} = L_1 - (L_2 - (L_3 - \cdots (L_{m-1} - L_m) \cdots))$ and $L_1 \supseteq L_2 \supseteq \cdots \supseteq L_{m-1} \supseteq L_m$ (this can be done as this is simply the "telescoping" normal form of the levels of the boolean hierarchy over $\Sigma_k^p$; see [5]). For $1 \leq \ell \leq m$, define $L'_\ell$ as the language accepted by the following $\Sigma_k^p$

machine: On input $x$, guess $x_1$ with $|x_1| \leq p(|x|)$, let $h(\langle x_1, x\rangle) = \langle y_1, y_2\rangle$, and accept if and only if $((x_1 \in \widehat{L}_{\Sigma^p_{k-1}} \Leftrightarrow y_1 \notin L_{\Delta^p_{k-1}}) \wedge y_2 \in L_\ell)$.

Note that $L'_\ell \in \Sigma^p_k$ for each $\ell$, and that $L'_1 \supseteq L'_2 \supseteq \cdots \supseteq L'_{m-1} \supseteq L'_m$. We will show that if $x$ is an easy string for length $p(|x|)$, then $x \in \overline{L_{\mathrm{DIFF}_m(\Sigma^p_k)}}$ if and only if $x \in L'_1 - (L'_2 - \cdots (L'_{m-1} - L'_m)\cdots)$.

So suppose that $x$ is an easy string for length $p(|x|)$. Define $\ell'$ to be the unique integer such that (a) $0 \leq \ell' \leq m$, (b) $x \in L'_s$ for $1 \leq s \leq \ell'$, and (c) $x \notin L'_s$ for $s > \ell'$. It is immediate that $x \in L'_1 - (L'_2 - \cdots (L'_{m-1} - L'_m)\cdots)$ if and only if $\ell'$ is odd.

Let $w$ be some string such that

$$(\exists x_1 \in (\Sigma^*)^{\leq p(|x|)})(\exists y_1)[h(\langle x_1, x\rangle) = \langle y_1, w\rangle \wedge (x_1 \in \widehat{L}_{\Sigma^p_{k-1}} \Leftrightarrow y_1 \notin L_{\Delta^p_{k-1}})],$$

and $w \in L_{\ell'}$ if $\ell' > 0$ ($\ell'$ here is the $\ell'$ already set in the previous paragraph). Note that such a $w$ exists, since $x$ is easy for length $p(|x|)$ and, by our definition of $\ell'$, $x \in L'_{\ell'}$. By the definition of $\ell'$ (namely, since $x \notin L'_s$ for $s > \ell'$), $w \notin L_s$ for all $s > \ell'$. It follows that $w \in L_{\mathrm{DIFF}_m(\Sigma^p_k)}$ if and only if $\ell'$ is odd. It is clear, keeping in mind the definition of $h$, that $(x \in \overline{L_{\mathrm{DIFF}_m(\Sigma^p_k)}} \Leftrightarrow w \in L_{\mathrm{DIFF}_m(\Sigma^p_k)})$, $(w \in L_{\mathrm{DIFF}_m(\Sigma^p_k)} \Leftrightarrow \ell'$ is odd), and ($\ell'$ is odd $\Leftrightarrow x \in L'_1 - (L'_2 - \cdots (L'_{m-1} - L'_m)\cdots)$). So $x \in \overline{L_{\mathrm{DIFF}_m(\Sigma^p_k)}} \Leftrightarrow x \in L'_1 - (L'_2 - \cdots (L'_{m-1} - L'_m)\cdots))$.

This completes the case where $x$ is easy, as $L'_1 - (L'_2 - \cdots (L'_{m-1} - L'_m)\cdots)$ in effect specifies a $\mathrm{DIFF}_m(\Sigma^p_k)$ algorithm.

We say that $x$ is *hard for length* $n$ if $|x| \leq n$ and $x$ is not easy for length $n$, i.e., if $|x| \leq n$ and for all $x_1$ with $|x_1| \leq n$, $(x_1 \in \widehat{L}_{\Sigma^p_{k-1}} \Leftrightarrow y_1 \in L_{\Delta^p_{k-1}})$, where $h(\langle x_1, x\rangle) = \langle y_1, y_2\rangle$. Note that if $x$ is hard for $p(|x|)$, then $x \notin L'_1$.

If $x$ is a hard string for length $p(|x|)$, then $x$ induces a many-one reduction from $(\widehat{L}_{\Sigma^p_{k-1}})^{\leq p(|x|)}$ to $L_{\Delta^p_{k-1}}$, namely, $\lambda x_1.f(x, x_1)$, where $f(x, x_1) = y_1$, where $y_1$ is the unique string such that $(\exists y_2)[h(\langle x_1, x\rangle) = \langle y_1, y_2\rangle]$. We will write $f_x$ for $\lambda x_1.f(x, x_1)$. Note that $f$ is computable in polynomial time. (This in turn certainly implies the true but not too relevant fact that, for each fixed $x$, $f_x$ is computable in polynomial time.)

So it is not hard to see that if we choose $p$ appropriately large, then a hard string $x$ for length $p(|x|)$ induces $\Sigma^p_{k-1}$ algorithms for $(L_1)^{=|x|}, (L_2)^{=|x|}, \ldots, (L_m)^{=|x|}$ (essentially since each is in $\Sigma^p_k = \mathrm{NP}^{\Sigma^p_{k-1}}$, $\widehat{L}_{\Sigma^p_{k-1}}$ is $\leq^p_{\mathrm{m}}$-complete for $\Sigma^p_{k-1}$, and $\mathrm{NP}^{\Delta^p_{k-1}} = \Sigma^p_{k-1}$), which we can use to obtain a $\mathrm{DIFF}_m(\Sigma^p_{k-1})$ algorithm for $(L_{\mathrm{DIFF}_m(\Sigma^p_k)})^{=|x|}$ and thus certainly a $\mathrm{DIFF}_m(\Sigma^p_k)$ algorithm for $\left(\overline{L_{\mathrm{DIFF}_m(\Sigma^p_k)}}\right)^{=|x|}$.

However, there is a problem. The problem is that we cannot combine the $\mathrm{DIFF}_m(\Sigma^p_k)$ algorithms for easy and hard strings into one $\mathrm{DIFF}_m(\Sigma^p_k)$ algorithm for $\overline{L_{\mathrm{DIFF}_m(\Sigma^p_k)}}$ that works for all strings. The reason why is that it is too difficult to decide whether a string is easy or hard: To decide this deterministically takes one query to $\Sigma^p_k$, and we cannot do that in a $\mathrm{DIFF}_m(\Sigma^p_k)$ algorithm. This is also the reason why the methods from [14] failed to prove that if $\mathrm{P} \bigoplus \Sigma^p_2 = \mathrm{NP} \bigoplus \Sigma^p_2$, then $\Sigma^p_2 = \Pi^p_2$. Recall from the introduction that the latter theorem was proven by Buhrman and Fortnow [4, p. 184]. We will generalize their technique at this point. In particular, the following lemma, which we will prove after we have finished the proof of this theorem, establishes a generalized version of the technique from [4]. It has been extended to deal with arbitrary levels of the polynomial hierarchy and to be useful in settings involving boolean hierarchies.

LEMMA 3.6. *Let $k > 1$. For all $L \in \Sigma_k^p$, there exist a polynomial $q$ and a set $\widehat{L} \in \Pi_{k-1}^p$ such that*

1. *for each natural number $n'$, $q(n') \geq n'$,*
2. *$\widehat{L} \subseteq \overline{L}$, and*
3. *if $x$ is hard for length $q(|x|)$, then $(x \in \overline{L} \Leftrightarrow x \in \widehat{L})$.*

We defer the proof of Lemma 3.6 and first finish the current proof. From Lemma 3.6, it follows that there exist sets $\widehat{L_1}, \widehat{L_2}, \ldots, \widehat{L_m} \in \Pi_{k-1}^p$ and polynomials $q_1, q_2, \ldots, q_m$ with the following properties for all $1 \leq \ell \leq m$:

1. $\widehat{L_\ell} \subseteq \overline{L_\ell}$, and
2. if $x$ is hard for length $q_\ell(|x|)$, then $(x \in \overline{L_\ell} \Leftrightarrow x \in \widehat{L_\ell})$.

Take $p$ to be an (easy-to-compute—we may without loss of generality require that there is a $t$ such that it is of the form $n^t + t$) polynomial such that $p$ is at least as large as all the $q_\ell$s, i.e., such that, for each natural number $n'$, we have $p(n') \geq \max\{q_1(n'), \ldots, q_m(n')\}$. By the definition of hardness and condition 1 of Lemma 3.6, if $x$ is hard for length $p(|x|)$, then $x$ is hard for length $q_\ell(|x|)$ for all $1 \leq \ell \leq m$. As promised earlier, we have now specified $p$. Define $\widehat{L}_{\mathrm{DIFF}_m(\Sigma_k^p)}$ as follows: On input $x$, guess $\ell$, $\ell$ even, $0 \leq \ell \leq m$, and accept if and only if both (a) $x \in L_\ell$ or $\ell = 0$, and (b) if $\ell < m$, then $x \in \widehat{L_{\ell+1}}$. Clearly, $\widehat{L}_{\mathrm{DIFF}_m(\Sigma_k^p)} \in \Sigma_k^p$. In addition, this set inherits certain properties from the $\widehat{L_\ell}$s. In particular, in light of the definition of $\widehat{L}_{\mathrm{DIFF}_m(\Sigma_k^p)}$, the definitions of the $\widehat{L_\ell}$s, and the fact that

$$x \in \overline{L_{\mathrm{DIFF}_m(\Sigma_k^p)}} \Leftrightarrow (\exists \ell, 0 \leq \ell \leq m, \ell \text{ even})[(\ell \neq 0 \Rightarrow x \in L_\ell) \wedge (\ell \neq m \Rightarrow x \in \overline{L_{\ell+1}})],$$

we have that the following properties hold:

1. $\widehat{L}_{\mathrm{DIFF}_m(\Sigma_k^p)} \subseteq \overline{L_{\mathrm{DIFF}_m(\Sigma_k^p)}}$, and
2. if $x$ is hard for length $p(|x|)$, then $(x \in \overline{L_{\mathrm{DIFF}_m(\Sigma_k^p)}} \Leftrightarrow x \in \widehat{L}_{\mathrm{DIFF}_m(\Sigma_k^p)})$.

Finally, we are ready to give the algorithm. Recall that $L'_1, L'_2, \ldots, L'_m$ are sets in $\Sigma_k^p$ such that (1) $L'_1 \supseteq L'_2 \supseteq \cdots \supseteq L'_{m-1} \supseteq L'_m$, and (2) if $x$ is easy for length $p(|x|)$, then $x \in \overline{L_{\mathrm{DIFF}_m(\Sigma_k^p)}}$ if and only if $x \in L'_1 - (L'_2 - (L'_3 - \cdots (L'_{m-1} - L'_m) \cdots))$, and (3) if $x$ is hard for length $p(|x|)$, then $x \notin L'_1$. We claim that for all $x$, $(x \in \overline{L_{\mathrm{DIFF}_m(\Sigma_k^p)}} \Leftrightarrow x \in (L'_1 \cup \widehat{L}_{\mathrm{DIFF}_m(\Sigma_k^p)}) - (L'_2 - (L'_3 - \cdots (L'_{m-1} - L'_m) \cdots)))$, which completes the proof of Theorem 3.4, as $\Sigma_k^p$ is closed under union.

($\Rightarrow$): Let $x \in \overline{L_{\mathrm{DIFF}_m(\Sigma_k^p)}}$. If $x$ is easy for length $p(|x|)$, then $x \in L'_1 - (L'_2 - (L'_3 - \cdots (L'_{m-1} - L'_m) \cdots))$, and so certainly $x \in (L'_1 \cup \widehat{L}_{\mathrm{DIFF}_m(\Sigma_k^p)}) - (L'_2 - (L'_3 - \cdots (L'_{m-1} - L'_m) \cdots))$. If $x$ is hard for length $p(|x|)$, then $x \in \widehat{L}_{\mathrm{DIFF}_m(\Sigma_k^p)}$ and $x \notin L'_\ell$ for all $\ell$ (since $x \notin L'_1$ and $L'_1 \supseteq L'_2 \supseteq \cdots \supseteq L'_m$). Thus, $x \in (L'_1 \cup \widehat{L}_{\mathrm{DIFF}_m(\Sigma_k^p)}) - (L'_2 - (L'_3 - \cdots (L'_{m-1} - L'_m) \cdots))$.

($\Leftarrow$): Suppose $x \in (L'_1 \cup \widehat{L}_{\mathrm{DIFF}_m(\Sigma_k^p)}) - (L'_2 - (L'_3 - \cdots (L'_{m-1} - L'_m) \cdots))$. If $x \in \widehat{L}_{\mathrm{DIFF}_m(\Sigma_k^p)}$, then $x \in \overline{L_{\mathrm{DIFF}_m(\Sigma_k^p)}}$. If $x \notin \widehat{L}_{\mathrm{DIFF}_m(\Sigma_k^p)}$, then $x \in L'_1 - (L'_2 - (L'_3 - \cdots (L'_{m-1} - L'_m) \cdots))$ and so $x$ must be easy for length $p(|x|)$ (as $x \in L'_1$, and this is possible only if $x$ is easy for length $p(|x|)$). However, this says that $x \in \overline{L_{\mathrm{DIFF}_m(\Sigma_k^p)}}$. $\square$

Having completed the proof of Theorem 3.4, we now return to the deferred proof of the lemma used within that theorem's proof.

*Proof of Lemma* 3.6. Let $L \in \Sigma_k^p$. We need to show that there exist a polynomial $q$ and a set $\widehat{L} \in \Pi_{k-1}^p$ such that

1. for each natural number $n'$, $q(n') \geq n'$,
2. $\widehat{L} \subseteq \overline{L}$, and
3. if $x$ is hard for length $q(|x|)$, then $(x \in \overline{L} \Leftrightarrow x \in \widehat{L})$.

From Fact 3.5, we know that $L'_{\Sigma_k^p}$ is $\leq_{\rm m}^p$-complete for $\Sigma_k^p$, $\widehat{L}_{\Sigma_{k-1}^p} \in \Sigma_{k-1}^p$, $L''_{\Sigma_{k-2}^p} \in \Sigma_{k-2}^p$, and

1. $L'_{\Sigma_k^p} = \{x \mid (\exists y \in \Sigma^{|x|})(\forall z \in \Sigma^{|x|})[\langle x, y, z\rangle \in L''_{\Sigma_{k-2}^p}]\}$, and
2. $\widehat{L}_{\Sigma_{k-1}^p} = \{\langle x, y, z\rangle \mid |x| = |y| \wedge (\exists z')[(|x| = |zz'|) \wedge \langle x, y, zz'\rangle \notin L''_{\Sigma_{k-2}^p}]\}$.

Note that $\overline{L'_{\Sigma_k^p}} = \{x \mid (\forall y \in \Sigma^{|x|})(\exists z \in \Sigma^{|x|})[\langle x, y, z\rangle \notin L''_{\Sigma_{k-2}^p}]\}$.

Since $L \in \Sigma_k^p$, and $L'_{\Sigma_k^p}$ is $\leq_{\rm m}^p$-complete for $\Sigma_k^p$, there exists a polynomial-time computable function $g$ such that, for all $x$, $(x \in L \Leftrightarrow g(x) \in L'_{\Sigma_k^p})$.

Let $q$ be such that (a) $(\forall x \in \Sigma^n)(\forall y \in \Sigma^{|g(x)|})(\forall z \in (\Sigma^*)^{\leq |g(x)|})[q(n) \geq |\langle g(x), y, z\rangle|]$ and (b) $(\forall \widehat{m} \geq 0)[q(\widehat{m}+1) > q(\widehat{m}) > 0]$. Note that we have ensured that for each natural number $n'$, $q(n') \geq n'$.

If $x$ is a hard string for length $p(|x|)$, then $x$ induces a many-one reduction from $(\widehat{L}_{\Sigma_{k-1}^p})^{\leq p(|x|)}$ to $L_{\Delta_{k-1}^p}$, namely, $\lambda x_1.f(x, x_1)$, where $f(x, x_1) = y_1$, where $y_1$ is the unique string such that $(\exists y_2)[h(\langle x_1, x\rangle) = \langle y_1, y_2\rangle]$. (This is the $h$ from the proof of Theorem 3.4. One should treat the current proof as if it occurs immediately after the statement of Lemma 3.6.) We will write $f_x$ for $\lambda x_1.f(x, x_1)$. Note that $f$ is computable in polynomial time. (This in turn certainly implies the true but not too relevant fact that, for each fixed $x$, $f_x$ is computable in polynomial time.)

Let $\widehat{L}$ be the language accepted by the following $\Pi_{k-1}^p$ machine.[2]

```
On input $x$:
        Compute $g(x)$
        Guess $y$ such that $|y| = |g(x)|$
        Set $w = \epsilon$ (i.e., the empty string)
        While $|w| < |g(x)|$
              if the $\Delta_{k-1}^p$ algorithm for $\widehat{L}_{\Sigma_{k-1}^p}$ induced by $x$ accepts $\langle g(x), y, w0\rangle$
              (that is, if $f_x(\langle g(x), y, w0\rangle) \in L_{\Delta_{k-1}^p}$),
              then $w = w0$
              else $w = w1$
        Accept if and only if $\langle g(x), y, w\rangle \notin L''_{\Sigma_{k-2}^p}$.
```

It remains to show that $\widehat{L}$ thus defined fulfills the properties of Lemma 3.6. First, note that the machine described above is clearly a $\Pi_{k-1}^p$ machine. To show that $\widehat{L} \subseteq \overline{L}$, suppose that $x \in \widehat{L}$. Then (keeping in mind the comments of footnote 2) for every $y \in \Sigma^{|g(x)|}$, there exists a string $w \in \Sigma^{|g(x)|}$ such that $\langle g(x), y, w\rangle \notin L''_{\Sigma_{k-2}^p}$. This implies that $g(x) \in \overline{L'_{\Sigma_k^p}}$ and thus that $x \in \overline{L}$.

Finally, suppose that $x$ is hard for length $q(|x|)$ and that $x \in \overline{L}$. We have to show that $x \in \widehat{L}$. Since $x \in \overline{L}$, $g(x) \in \overline{L'_{\Sigma_k^p}}$. So, $(\forall y \in \Sigma^{|g(x)|})(\exists z \in \Sigma^{|g(x)|})[\langle g(x), y, z\rangle \notin L''_{\Sigma_{k-2}^p}]$. Since $x$ is hard for length $q(|x|)$, $(\forall y \in \Sigma^{|g(x)|})(\forall w \in (\Sigma^*)^{\leq |g(x)|})[\langle g(x), y, w\rangle \in \widehat{L}_{\Sigma_{k-1}^p} \Leftrightarrow f_x(\langle g(x), y, w\rangle) \in L_{\Delta_{k-1}^p}]$. It follows that the algorithm above will find, for

[2]For $k > 1$, $\Pi_{k-1}^p = \mathrm{coNP}^{\Sigma_{k-2}^p}$, and by a $\Pi_{k-1}^p$ machine we mean, for the duration of this proof, a co-nondeterministic machine with a $\Sigma_{k-2}^p$ oracle. A co-nondeterministic machine by definition accepts if and only if *all* of its computation paths are accepting paths.

every $y \in \Sigma^{|g(x)|}$, a witness $w$ such that $\langle g(x), y, w\rangle \notin L''_{\Sigma^p_{k-2}}$, and thus the algorithm will accept $x$. $\square$

**4. Downward collapse from closure under complementation.** This section's main result is the following theorem.

THEOREM 4.1. *Let $s, m > 0$ and $0 < i < k-1$. If $\mathrm{DIFF}_s(\Sigma^p_i) \bigoplus \mathrm{DIFF}_m(\Sigma^p_k)$ is closed under complementation, then $\mathrm{DIFF}_m(\Sigma^p_k) = \mathrm{coDIFF}_m(\Sigma^p_k)$.*

The $s = 1$ case of Theorem 4.1 is as follows: If $m > 0$, $0 < i < k-1$, and $\Sigma^p_i \bigoplus \mathrm{DIFF}_m(\Sigma^p_k)$ is closed under complementation, then $\mathrm{DIFF}_m(\Sigma^p_k) = \mathrm{coDIFF}_m(\Sigma^p_k)$. In the present paper, this $s = 1$ case is used, along with Theorem 3.4, to establish Theorem 3.3.

Theorem 4.1 is also of interest in its own right as a reflection of how closure under complementation of even quite general symmetric difference classes implies a downward collapse. Selivanov [25, 26] shows that if certain symmetric difference classes are closed under complementation, then the polynomial hierarchy collapses. His result is, however, very different than this section's main result, Theorem 4.1, as Selivanov collapses the polynomial hierarchy to a higher level, and thus shows merely an upward translation of equality. In contrast, our Theorem 4.1 collapses the difference hierarchy over $\Sigma^p_k$ to a level that is contained in the classes of its complementation hypothesis—thus obtaining a *downward* translation of equality. Also, we note that Theorem 4.1 implies a collapse of the polynomial hierarchy to a class a full level lower in the difference hierarchy over $\Sigma^p_{k+1}$ than could be concluded without our downward collapse result (namely a collapse to $\mathrm{DIFF}_m(\Sigma^p_k) \bigoplus \mathrm{DIFF}_{m-1}(\Sigma^p_{k+1})$, in light of the strongest known "Boolean Hierarchy/Polynomial Hierarchy-collapse connection," see [13, Theorem 5.1], [23, Corollary 27], and the related discussion in section 5).

Before proving Theorem 4.1, we fix some useful complete languages. Throughout the paper we will use the names introduced in this fact for these sets (or more specifically, let us consider, for each $i$, the three "Sigma"-type sets of the form guaranteed by the fact to be fixed, and let us attach to them the names used in the fact for those sets, and then based on those apply the second half of the fact to assign fixed sets to be the $L_{\Pi^p_i}$'s and other "Pi"-type sets).

FACT 4.2. *For each $i \geq 1$, there exist a set $L_{\Sigma^p_i}$ that is $\leq^p_{\mathrm{m}}$-complete for $\Sigma^p_i$, a set $\widetilde{L}_{\Sigma^p_{i+1}}$ that is $\leq^p_{\mathrm{m}}$-complete for $\Sigma^p_{i+1}$, and a set $L^\dagger_{\Sigma^p_{i+2}}$ that is $\leq^p_{\mathrm{m}}$-complete for $\Sigma^p_{i+2}$, such that these sets satisfy*

$$\widetilde{L}_{\Sigma^p_{i+1}} = \{x \mid (\exists y \in \Sigma^{|x|})[\langle x, y\rangle \notin L_{\Sigma^p_i}]\}$$

*and*

$$L^\dagger_{\Sigma^p_{i+2}} = \{x \mid (\exists y \in \Sigma^{|x|})[\langle x, y\rangle \notin \widetilde{L}_{\Sigma^p_{i+1}}]\}.$$

*For each $i \geq 1$, let $L_{\Pi^p_i} = \overline{L_{\Sigma^p_i}}$ and define $L_{\mathrm{DIFF}_1(\Pi^p_i)} = L_{\Pi^p_i}$ and for all $j \geq 2$, $L_{\mathrm{DIFF}_j(\Pi^p_i)} = \{\langle x, y\rangle \mid x \in L_{\Pi^p_i} \wedge y \notin L_{\mathrm{DIFF}_{j-1}(\Pi^p_i)}\}$. It is not hard to see that $L_{\mathrm{DIFF}_j(\Pi^p_i)}$ is many-one complete for $\mathrm{DIFF}_j(\Pi^p_i)$ for all $j \geq 1$. Note also that $\mathrm{DIFF}_j(\Pi^p_i) = \mathrm{DIFF}_j(\Sigma^p_i)$ if $j$ is even and $\mathrm{DIFF}_j(\Pi^p_i) = \mathrm{co}\overline{\mathrm{DIFF}_j(\Sigma^p_i)}$ if $j$ is odd. Let $L_{\mathrm{DIFF}_s(\Sigma^p_i)} = L_{\mathrm{DIFF}_s(\Pi^p_i)}$ if $s$ is even and $L_{\mathrm{DIFF}_s(\Sigma^p_i)} = \overline{L_{\mathrm{DIFF}_s(\Pi^p_i)}}$ if $s$ is odd. Then $L_{\mathrm{DIFF}_s(\Sigma^p_i)}$ is $\leq^p_{\mathrm{m}}$-complete for $\mathrm{DIFF}_s(\Sigma^p_i)$.*

The reason these sets exist is similar to the reason that the sets of Fact 3.5 exist, but may at first seem a bit confusing, due to the fact that in Fact 4.2 $\widetilde{L}_{\Sigma^p_{i+1}}$ is being treated as a set of strings in its own definition but as a set of pairs of strings in the definition of $L^{\dagger}_{\Sigma^p_{i+2}}$. However, since the pairing function maps strings to strings, this isn't a problem; it merely requires some pairing in forming the sets. For example, to give the intuition of what is going on, consider the following sets:

- $A_{\Sigma^p_0} = \{\langle\langle F, v\rangle, w\rangle \mid v$ specifies assignments to the first half of $F$'s variables and $w$ specifies assignments to the second half of $F$'s variables and $F$ under the (complete) assignment specified by $v$ and $w$ evaluates to false$\}$.
- $A_{\Sigma^p_1} = \{\langle F, v\rangle \mid$ $v$ specifies assignments to the first half of $F$'s variables and there exists a $w$ specifying assignments to the second half of $F$'s variables such that $\langle\langle F, v\rangle, w\rangle \notin A_{\Sigma^p_0}\}$.
- $A_{\Sigma^p_2} = \{F \mid$ there exists a $v$ specifying assignments to the first half of $F$'s variables such that $\langle F, v\rangle \notin A_{\Sigma^p_1}\}$.

These sets are easily seen to be, respectively, $\leq^p_m$-complete for $\Sigma^p_0$ (trivial, in this case), $\Sigma^p_1$, and $\Sigma^p_2$. Caveat: These are not exactly the sets of Fact 4.2, as here we have not been careful to make sure the quantified lengths are exactly the same length as the input, and we have been sloppy about what "half" means when the number of variables is odd; however, this example should make it clearer that sets satisfying Fact 4.2 exist.

*Proof of Theorem* 4.1. Let $s, m > 0$ and $0 < i < k-1$. $L_{\mathrm{DIFF}_s(\Sigma^p_i)} \tilde{\oplus} L_{\mathrm{DIFF}_m(\Sigma^p_k)}$ is $\leq^p_m$-hard for $\mathrm{DIFF}_s(\Sigma^p_i) \bigoplus \mathrm{DIFF}_m(\Sigma^p_k)$ by Lemma 3.2 and $L_{\mathrm{DIFF}_s(\Sigma^p_i)} \tilde{\oplus} L_{\mathrm{DIFF}_m(\Sigma^p_k)}$ is clearly in $\mathrm{DIFF}_s(\Sigma^p_i) \bigoplus \mathrm{DIFF}_m(\Sigma^p_k)$. Since by assumption $\mathrm{DIFF}_s(\Sigma^p_i) \bigoplus \mathrm{DIFF}_m(\Sigma^p_k)$ is closed under complementation, there exists a (polynomial-time) many-one reduction $h$ from $L_{\mathrm{DIFF}_s(\Sigma^p_i)} \tilde{\oplus} L_{\mathrm{DIFF}_m(\Sigma^p_k)}$ to its complement. That is, for all $x_1, x_2, y_1, y_2 \in \Sigma^*$: If $h(\langle x_1, x_2\rangle) = \langle y_1, y_2\rangle$, then $(\langle x_1, x_2\rangle \in L_{\mathrm{DIFF}_s(\Sigma^p_i)} \tilde{\oplus} L_{\mathrm{DIFF}_m(\Sigma^p_k)} \Leftrightarrow \langle y_1, y_2\rangle \notin L_{\mathrm{DIFF}_s(\Sigma^p_i)} \tilde{\oplus} L_{\mathrm{DIFF}_m(\Sigma^p_k)})$. Equivalently, for all $x_1, x_2, y_1, y_2 \in \Sigma^*$:

FACT 1: *If* $h(\langle x_1, x_2\rangle) = \langle y_1, y_2\rangle$, *then*: $(x_1 \in L_{\mathrm{DIFF}_s(\Sigma^p_i)} \Leftrightarrow x_2 \notin L_{\mathrm{DIFF}_m(\Sigma^p_k)})$ *if and only if* $(y_1 \in L_{\mathrm{DIFF}_s(\Sigma^p_i)} \Leftrightarrow y_2 \in L_{\mathrm{DIFF}_m(\Sigma^p_k)})$.

We can use $h$ to recognize some of $\overline{L_{\mathrm{DIFF}_m(\Sigma^p_k)}}$ by a $\mathrm{DIFF}_m(\Sigma^p_k)$ algorithm. In particular, we say that a string $x$ is *easy for length* $n$ if there exists a string $x_1$ such that $|x_1| \leq n$ and $(x_1 \in L_{\mathrm{DIFF}_s(\Sigma^p_i)} \Leftrightarrow y_1 \in L_{\mathrm{DIFF}_s(\Sigma^p_i)})$, where $h(\langle x_1, x\rangle) = \langle y_1, y_2\rangle$.

Let $p$ be a fixed polynomial, which will be exactly specified later in the proof. We have the following algorithm to test whether $x \in \overline{L_{\mathrm{DIFF}_m(\Sigma^p_k)}}$ in the case that (our input) $x$ is an easy string for length $p(|x|)$. On input $x$, guess $x_1$ with $|x_1| \leq p(|x|)$, let $h(\langle x_1, x\rangle) = \langle y_1, y_2\rangle$, and accept if and only if $((x_1 \in L_{\mathrm{DIFF}_s(\Sigma^p_i)} \Leftrightarrow y_1 \in L_{\mathrm{DIFF}_s(\Sigma^p_i)}) \wedge y_2 \in L_{\mathrm{DIFF}_m(\Sigma^p_k)})$. This algorithm is not necessarily a $\mathrm{DIFF}_m(\Sigma^p_k)$ algorithm, but in the same way as in the proof of Theorem 3.4, we can construct sets $L'_1, L'_2, \ldots, L'_m \in \Sigma^p_k$ such that if $x$ is an easy string for length $p(|x|)$, then $x \in \overline{L_{\mathrm{DIFF}_m(\Sigma^p_k)}}$ if and only if $x \in L'_1 - (L'_2 - (L'_3 - \cdots (L'_{m-1} - L'_m)\cdots))$.

We say that $x$ is *hard for length* $n$ if $|x| \leq n$ and $x$ is not easy for length $n$, i.e., if $|x| \leq n$ and, for all $x_1$ with $|x_1| \leq n$, $(x_1 \in L_{\mathrm{DIFF}_s(\Sigma^p_i)} \Leftrightarrow y_1 \notin L_{\mathrm{DIFF}_s(\Sigma^p_i)})$, where $h(\langle x_1, x\rangle) = \langle y_1, y_2\rangle$.

If $x$ is a hard string for length $n$, then $x$ induces a many-one reduction from $(L_{\mathrm{DIFF}_s(\Sigma^p_i)})^{\leq n}$ to $\overline{L_{\mathrm{DIFF}_s(\Sigma^p_i)}}$, namely, $\lambda x_1.f(x, x_1)$, where $f(x, x_1) = y_1$, where $y_1$ is the unique string such that $(\exists y_2)[h(\langle x_1, x\rangle) = \langle y_1, y_2\rangle]$. We will write $f_x$ for $\lambda x_1.f(x, x_1)$. Note that $f$ is computable in polynomial time. (This in turn certainly implies the true but not too relevant fact that, for each fixed $x$, $f_x$ is computable in polynomial time.)

It is known that a collapse of the boolean hierarchy over $\Sigma_i^p$ implies a collapse of the polynomial hierarchy. A long series of papers studied the question to what level the polynomial hierarchy collapses in that case. The best known results (e.g., [8, 3, 14, 23, 13], see, especially, the strongest such connection, which is that obtained independently in [23] and [13]) conclude a collapse of the polynomial hierarchy to a level within the boolean hierarchy over $\Sigma_{i+1}^p$. Though a hard string for length $n$ only induces a many-one reduction between initial segments of $L_{\mathrm{DIFF}_s(\Sigma_i^p)}$ and $\overline{L_{\mathrm{DIFF}_s(\Sigma_i^p)}}$, we would nevertheless like to derive at least a $\mathrm{P}^{\Sigma_{k-1}^p}$ algorithm for some of $L^{\dagger}_{\Sigma_{i+2}^p}$. The following lemma does exactly that.

LEMMA 4.3. *Let $s, m > 0$ and $0 < i < k-1$, and suppose that* $\mathrm{DIFF}_s(\Sigma_i^p) \bigoplus \mathrm{DIFF}_m(\Sigma_k^p) = \mathrm{co}(\mathrm{DIFF}_s(\Sigma_i^p) \bigoplus \mathrm{DIFF}_m(\Sigma_k^p))$. *There exist a set $D \in \mathrm{P}^{\Sigma_{i+1}^p}$ and a polynomial $r$ such that for all $n$,* (a) *$r(n+1) > r(n) > 0$ and* (b) *for all $x \in \Sigma^*$, if $x$ is a hard string for length $r(n)$ then for all $y \in (\Sigma^*)^{\leq n}$,*

$$y \in L^{\dagger}_{\Sigma_{i+2}^p} \Leftrightarrow \langle x, 1^n, y\rangle \in D.$$

We defer the proof of Lemma 4.3 and first finish the proof of the current theorem.

We will use the result of Lemma 4.3 to obtain a $\mathrm{P}^{\Sigma_{k-1}^p}$ algorithm that for any string $x$ that is hard for length $p(|x|)$ will determine whether $x \in L_{\mathrm{DIFF}_m(\Sigma_k^p)}$.

Let $L_1, L_2, \ldots, L_m$ be languages in $\Sigma_k^p$ such that $L_{\mathrm{DIFF}_m(\Sigma_k^p)} = L_1 - (L_2 - (L_3 - \cdots (L_{m-1} - L_m) \cdots))$. Since $L_{\Sigma_k^p}$ is complete for $\Sigma_k^p$, there exist functions $g_1, \ldots, g_m$ that many-one reduce $L_1, \ldots, L_m$ to $L_{\Sigma_k^p}$, respectively. Let the output sizes of all the $g_j$'s be bounded by the polynomial $p'$, which without loss of generality satisfies $(\forall \widehat{m} \geq 0)[p'(\widehat{m}+1) > p'(\widehat{m}) > 0]$. So there exists a polynomial-time machine that queries strings of length at most $p'(n)$ on inputs of length $n$ to $L_{\Sigma_k^p}$ and that accepts $L_{\mathrm{DIFF}_m(\Sigma_k^p)}$.

A $\Sigma_0^p$ machine is an oracle P machine; for $z \geq 1$, a $\Sigma_z^p$ machine is a polynomial-time-bounded $z$-alternation-block-bounded oracle machine with the first alternation block existential. For example, the class of languages accepted by $\Sigma_2^p$ machines allowed $\Sigma_3^p$ oracles is $\Sigma_5^p$. Let $M$ be a $\Sigma_{k-(i+2)}^p$ machine recognizing $L_{\Sigma_k^p}$ with oracle queries to $L^{\dagger}_{\Sigma_{i+2}^p}$ and running in time $q'$ for some polynomial $q'$ satisfying $(\forall \widehat{m} \geq 0)[q'(\widehat{m}+1) > q'(\widehat{m}) > 0]$. Let $p$ be an easily computable polynomial satisfying $(\forall \widehat{m} \geq 0)[p(\widehat{m}+1) > p(\widehat{m}) > 0]$ and for all $n$, $p(n) \geq r(q'(p'(n)))$, where $r$ is the polynomial of Lemma 4.3. As promised, we now have specified $p$.

If $x$ is a hard string for length $p(|x|)$, then $x$ is also a hard string for length $r(q'(p'(|x|)))$. So, by Lemma 4.3, for all $y \in (\Sigma^*)^{\leq q'(p'(|x|))}$, $y \in L^{\dagger}_{\Sigma_{i+2}^p} \Leftrightarrow \langle x, 1^n, y\rangle \in D$. Define the following $\mathrm{P}^{\Sigma_{k-1}^p}$ set $E$: On input $\langle x, z\rangle$, simulate $M$ on input $z$ and replace every query $y$ to $L^{\dagger}_{\Sigma_{i+2}^p}$ by query $\langle x, 1^n, y\rangle$ to $D$. (Note that if $i < k-2$, $E$ is even in $\Sigma_{k-1}^p$.) Clearly, for all $x \in \Sigma^*$, if $x$ is a hard string for length $p(|x|)$, then for all $z \in (\Sigma^*)^{\leq p'(|x|)}$, $\langle x, z\rangle \in E$ if and only if $z \in L_{\Sigma_k^p}$.

Recall that there exists a polynomial-time machine that determines whether $x \in L_{\mathrm{DIFF}_m(\Sigma_k^p)}$ with oracle queries of length at most $p'(|x|)$ to $L_{\Sigma_k^p}$. If $x$ is a hard string for length $p(|x|)$, we can replace every query $z$ to $L_{\Sigma_k^p}$ by query $\langle x, z\rangle$ to $E$. We have now defined a $\mathrm{P}^{\mathrm{P}^{\Sigma_{k-1}^p}} = \mathrm{P}^{\Sigma_{k-1}^p}$ algorithm that for any string $x$ that is hard for length $p(|x|)$ will determine whether $x \in L_{\mathrm{DIFF}_m(\Sigma_k^p)}$.

However, now we have an outright $\mathrm{DIFF}_m(\Sigma_k^p)$ algorithm for $\overline{L_{\mathrm{DIFF}_m(\Sigma_k^p)}}$: For $1 \leq \ell \leq m$ define an $\mathrm{NP}^{\Sigma_{k-1}^p}$ machine $N_\ell$ as follows: On input $x$, the NP base machine of $N_\ell$ executes the following algorithm:

1. Using its $\Sigma_{k-1}^p$ oracle, it deterministically determines whether the input $x$ is an easy string for length $p(|x|)$. This can be done, as checking whether the input is an easy string for length $p(|x|)$ can be done by one query to $\Sigma_{i+1}^p$, and $i+1 \leq k-1$ by our $i < k-1$ hypothesis.
2. If the previous step determined that the input is not an easy string, then the input must be a hard string for length $p(|x|)$. If $\ell = 1$, then simulate the $\mathrm{P}^{\Sigma_{k-1}^p}$ algorithm for hard strings to determine whether $x \in L_{\mathrm{DIFF}_m(\Sigma_k^p)}$ and accept if and only if $x \notin L_{\mathrm{DIFF}_m(\Sigma_k^p)}$. If $\ell > 1$, then reject.
3. If the first step determined that the input $x$ is easy for length $p(|x|)$, then our NP machine simulates (using itself and its oracle) the $\Sigma_k^p$ algorithm for $L'_\ell$ on input $x$.

Note that the $\Sigma_{k-1}^p$ oracle in the above algorithm is being used for a number of different sets. However, as $\Sigma_{k-1}^p$ is closed under disjoint union, this presents no problem as we can use the disjoint union of the sets, while modifying the queries so they address the appropriate part of the disjoint union.

It follows that, for all $x$, $x \in \overline{L_{\mathrm{DIFF}_m(\Sigma_k^p)}}$ if and only if $x \in L(N_1) - (L(N_2) - (L(N_3) - \cdots (L(N_{m-1}) - L(N_m)) \cdots))$. Since $\overline{L_{\mathrm{DIFF}_m(\Sigma_k^p)}}$ is complete for $\mathrm{coDIFF}_m(\Sigma_k^p)$, it follows that $\mathrm{DIFF}_m(\Sigma_k^p) = \mathrm{coDIFF}_m(\Sigma_k^p)$. ☐

We now give the proof of Lemma 4.3. This proof should be seen in the context of the proof of Theorem 4.1 and Fact 4.2 as some notations we are going to use are defined there.

*Proof of Lemma* 4.3. Our proof generalizes a proof from [3]. Let $\langle \cdots \rangle$ be a pairing function that maps sequences of up to $2s+2$ of strings over $\Sigma^*$ to $\Sigma^*$ having the standard properties such as polynomial-time computability and invertibility, etc. Let $t$ be a polynomial such that $|\langle x_1, x_2, \ldots, x_j \rangle| \leq t(\max\{|x_1|, |x_2|, \ldots, |x_j|\})$ for all $1 \leq j \leq 2s+2$ and all $x_1, x_2, \ldots, x_j \in \Sigma^*$. Without loss of generality let $t$ be such that $t(n+1) > t(n) > 0$ for all $n$. Define

$$t^{(0)}(n) = n \text{ and } t^{(j)}(n) = \underbrace{t(t(\cdots t}_{j\ times}(n)\cdots))$$

for all $n$ and all $j \geq 1$.

Define $r'$ to be a polynomial such that $r'(n+1) > r'(n) > 0$ and $r'(n) \geq t^{(s-1)}(n)$ for all $n$. Let $n$ be an integer. Suppose that $x$ is a hard string for length $r'(n)$, where hardness is defined as in the proof of Theorem 4.1. Then (recall the sets fixed/named in Fact 4.2), for all $y$ such that $|y| \leq r'(n)$,

$$y \in L_{\mathrm{DIFF}_s(\Sigma_i^p)} \Leftrightarrow f_x(y) \notin L_{\mathrm{DIFF}_s(\Sigma_i^p)},$$

or equivalently

$$y \in L_{\mathrm{DIFF}_s(\Pi_i^p)} \Leftrightarrow f_x(y) \notin L_{\mathrm{DIFF}_s(\Pi_i^p)}.$$

Recall that $f_x = \lambda y.f(x,y)$ and that $f$ can be computed in polynomial time. If $s > 1$, let $y = \langle y_1, y_2 \rangle$ and let $f_x(y) = \langle z_1, z_2 \rangle$. Then, for all $y_1, y_2 \in \Sigma^*$ such that $|y_1| \leq n$ and $|y_2| \leq t^{(s-2)}(n)$,

$$y_1 \in L_{\Pi_i^p} \wedge y_2 \notin L_{\mathrm{DIFF}_{s-1}(\Pi_i^p)} \Leftrightarrow z_1 \notin L_{\Pi_i^p} \vee z_2 \in L_{\mathrm{DIFF}_{s-1}(\Pi_i^p)}. \tag{4.1}$$

If $s > 1$, we say that $y_1$ is $s$-easy for length $n$ if and only if $|y_1| \le n$ and $(\exists y_2\ |y_2| \le t^{(s-2)}(n))[z_1 \notin L_{\Pi_i^p}]$. $y_1$ is said to be $s$-hard for length $n$ if and only if $|y_1| \le n$, $y_1 \in L_{\Pi_i^p}$, and $(\forall y_2\ |y_2| \le t^{(s-2)}(n))[z_1 \in L_{\Pi_i^p}]$. Observe that the above notions are defined with respect to our hard string $x$, since $z_1$ depends on $x$, $y_1$, and $y_2$. Furthermore, according to (4.1), if $y_1$ is $s$-easy for length $n$ then $y_1 \in L_{\Pi_i^p}$.

Suppose there exists an $s$-hard string $\omega_s$ for length $n$. Let $f_{(x,\omega_s)}$ be the function defined by $f_x(\langle \omega_s, y\rangle) = \langle z_1, f_{(x,\omega_s)}(y)\rangle$. Note that there exists a polynomial-time computable function $f_2$ such that $f_{(x,\omega_s)} = \lambda y.f_2(x, \omega_s, y)$. If $s-1 > 1$, we define $(s-1)$-easy and $(s-1)$-hard strings in analogy to the above. If an $(s-1)$-hard string exists we can repeat the process and define $(s-2)$-easy and $(s-2)$-hard strings and so on. Note that the definition of $j$-easy and $j$-hard strings can only be made with respect to our hard string $x$, some fixed $s$-hard string $\omega_s$, some fixed $(s-1)$-hard string $\omega_{s-1}$, ..., some fixed $(j+1)$-hard string $\omega_{j+1}$. If we have found a sequence of strings $(\omega_s, \omega_{s-1}, \dots, \omega_2)$ (note that if $s = 1$, $(\omega_s, \omega_{s-1}, \dots, \omega_2)$ is the empty sequence) such that every $\omega_j$ is $j$-hard with respect to $(x, \omega_s, \omega_{s-1}, \dots, \omega_{j+1})$, then we have for all $y$, $|y| \le n$,

$$y \in L_{\Pi_i^p} \Leftrightarrow f_{(x,\omega_s,\omega_{s-1},\dots,\omega_2)}(y) \notin L_{\Pi_i^p}.$$

We say that a string $y$ is 1-easy for length $n$ if and only if it holds that $|y| \le n$ and $f_{(x,\omega_s,\omega_{s-1},\dots,\omega_2)}(y) \notin L_{\Pi_i^p}$. We define that no string is 1-hard for length $n$.

$(x)$ is called a hard sequence for length $n$ if and only if $x$ is hard for length $r'(n)$. A sequence $(x, \omega_s, \omega_{s-1}, \dots, \omega_j)$ of strings is called a hard sequence for length $n$ if and only if $x$ is hard for length $r'(n)$ and for all $\ell$, $j \le \ell \le s$, $\omega_\ell$ is $\ell$-hard for length $n$ with respect to $(x, \omega_s, \omega_{s-1}, \dots, \omega_{\ell+1})$. Note that given a hard sequence $(x, \omega_s, \omega_{s-1}, \dots, \omega_j)$ for length $n$, the strings in $(L_{\Pi_i^p})^{\le n}$ divide into $(j-1)$-easy and $(j-1)$-hard strings (with respect to $(x, \omega_s, \omega_{s-1}, \dots, \omega_j)$) for length $n$.

$(x)$ is called a maximal hard sequence for length $n$ if and only if $(x)$ is a hard sequence for length $n$ and there exists no $s$-hard string for length $n$. A hard sequence $(x, \omega_s, \omega_{s-1}, \dots, \omega_j)$ for length $n$ is called a maximal hard sequence for length $n$ if and only if there exists no $(j-1)$-hard string for length $n$ with respect to $(x, \omega_s, \omega_{s-1}, \dots, \omega_j)$. When in what follows we denote a maximal hard sequence by $(x, \omega_s, \omega_{s-1}, \dots, \omega_j)$, we implicitly include the case that the maximal hard sequence might be $(x)$ or $(x, \omega_s)$ or ....

*Claim* 1. There exists a set $A \in \Sigma_i^p$ such that if $(x, \omega_s, \omega_{s-1}, \dots, \omega_j)$ is a maximal hard sequence for length $n$, then for all $y$ and $n$ satisfying $|y| \le n$,

$$y \in L_{\Pi_i^p} \Leftrightarrow \langle x, 1^n, \omega_s, \omega_{s-1}, \dots, \omega_j, y\rangle \in A.$$

*Proof of Claim* 1. Let $(x, \omega_s, \omega_{s-1}, \dots, \omega_j)$ be a maximal hard sequence for length $n$. If $(x, \omega_s, \omega_{s-1}, \dots, \omega_j) = (x)$, we let $j = s+1$. Note that $j \ge 2$ and that the strings in $(L_{\Pi_i^p})^{\le n}$ are exactly the strings of length at most $n$ that are $(j-1)$-easy with respect to $(x, \omega_s, \omega_{s-1}, \dots, \omega_j)$. It is immediate from the definition that testing whether a string $y$ is $(j-1)$-easy for length $n$ with respect to $(x, \omega_s, \omega_{s-1}, \dots, \omega_j)$ can be done by a $\Sigma_i^p$ algorithm running in time polynomial in $n$: If $j \ge 3$, check that $|y| \le n$, guess $y_2$, $|y_2| \le t^{(j-3)}(n)$, compute $f_{(x,\omega_s,\omega_{s-1},\dots,\omega_j)}(\langle y, y_2\rangle) = \langle z_1, z_2\rangle$, and accept if and only if $z_1 \notin L_{\Pi_i^p}$. If $j = 2$, check $|y| \le n$, and accept if and only if $f_{(x,\omega_s,\omega_{s-1},\dots,\omega_2)}(y) \notin L_{\Pi_i^p}$.

*Claim* 2. There exist a set $B \in \Sigma_i^p$ and a polynomial $\widehat{p}$ such that $(\forall n \ge 0)[\widehat{p}(n+1) > \widehat{p}(n) > 0]$ and if $(x, \omega_s, \omega_{s-1}, \dots, \omega_j)$ is a maximal hard sequence for length $\widehat{p}(n)$, then for all $y$ and $n$ satisfying $|y| \leq n$,

$$y \in \widetilde{L}_{\Sigma^p_{i+1}} \Leftrightarrow \langle x, 1^{\widehat{p}(n)}, \omega_s, \omega_{s-1}, \ldots, \omega_j, y\rangle \in B.$$

*Proof of Claim* 2. Let $A \in \Sigma^p_i$ as in Claim 1. Let $y$ be a string such that $|y| \leq n$. According to the definition of $\widetilde{L}_{\Sigma^p_{i+1}}$,

$$y \in \widetilde{L}_{\Sigma^p_{i+1}} \Leftrightarrow (\exists z \in \Sigma^{|y|})[\langle y, z\rangle \notin L_{\Sigma^p_i}].$$

Recall that $L_{\Pi^p_i} = \overline{L_{\Sigma^p_i}}$ from Fact 4.2. Define $\widehat{p}$ to be a polynomial such that $\widehat{p}(n+1) > \widehat{p}(n) > 0$ and $\widehat{p}(n) \geq t(n)$ for all $n$. Applying Claim 1 we obtain that if $(x, \omega_s, \omega_{s-1}, \ldots, \omega_j)$ is a maximal hard sequence for length $\widehat{p}(n)$, then

$$y \in \widetilde{L}_{\Sigma^p_{i+1}} \Leftrightarrow (\exists z \in \Sigma^{|y|})[\langle x, 1^{\widehat{p}(n)}, \omega_s, \omega_{s-1}, \ldots, \omega_j, \langle y, z\rangle\rangle \in A].$$

We define $B$ to be the set $B = \{\langle x, 1^{\widehat{p}(n)}, \omega_s, \omega_{s-1}, \ldots, \omega_j, y\rangle \mid (\exists z \in \Sigma^{|y|})[\langle x, 1^{\widehat{p}(n)}, \omega_s, \omega_{s-1}, \ldots, \omega_j, \langle y, z\rangle\rangle \in A]\}$. Clearly $B \in \Sigma^p_i$. This proves Claim 2.

*Claim* 3. There exist a set $C \in \Sigma^p_{i+1}$ and a polynomial $\widehat{p}_1$ such that $(\forall n \geq 0)[\widehat{p}_1(n+1) > \widehat{p}_1(n) > 0]$ and if $(x, \omega_s, \omega_{s-1}, \ldots, \omega_\ell)$ is a maximal hard sequence for length $\widehat{p}_1(n)$, then for all $y$ and $n$ satisfying $|y| \leq n$,

$$y \in L^{\dagger}_{\Sigma^p_{i+2}} \Leftrightarrow \langle x, 1^n, \omega_s, \omega_{s-1}, \ldots, \omega_\ell, y\rangle \in C.$$

*Proof of Claim* 3. Let $B \in \Sigma^p_i$ and $\widehat{p}$ be a polynomial, both as defined in Claim 2. Let $y$ be a string such that $|y| \leq n$. According to the definition of $L^{\dagger}_{\Sigma^p_{i+2}}$,

$$y \in L^{\dagger}_{\Sigma^p_{i+2}} \Leftrightarrow (\exists z \in \Sigma^{|y|})[\langle y, z\rangle \notin \widetilde{L}_{\Sigma^p_{i+1}}].$$

Define $\widehat{p}_1$ to be a polynomial such that $\widehat{p}_1(n+1) > \widehat{p}_1(n) > 0$ and $\widehat{p}_1(n) \geq \widehat{p}(t(n))$ for all $n$. Applying Claim 2, we obtain that if $(x, \omega_s, \omega_{s-1}, \ldots, \omega_\ell)$ is a maximal hard sequence for length $\widehat{p}_1(n)$, then

$$y \in L^{\dagger}_{\Sigma^p_{i+2}} \Leftrightarrow (\exists z \in \Sigma^{|y|})[\langle x, 1^{\widehat{p_1}(n)}, \omega_s, \omega_{s-1}, \ldots, \omega_\ell, \langle y, z\rangle\rangle \notin B].$$

Let $C = \{\langle x, 1^n, \omega_s, \omega_{s-1}, \ldots, \omega_j, y\rangle \mid (\exists z \in \Sigma^{|y|})[\langle x, 1^{\widehat{p_1}(n)}, \omega_s, \omega_{s-1}, \ldots, \omega_j, \langle y, z\rangle\rangle \notin B]\}$. Clearly $C \in \Sigma^p_{i+1}$. This concludes the proof of Claim 3.

We are now ready to prove the claim of Lemma 4.3. Note that the set

$$\begin{aligned} E = \{\langle x, 1^n, \omega_s, \omega_{s-1}, \ldots, \omega_j\rangle \mid\ & \text{for all } \ell, j \leq \ell \leq s, \\ & \omega_\ell \text{ is } \ell\text{-hard for length } n \text{ with respect to } (x, \omega_s, \omega_{s-1}, \ldots, \omega_{\ell+1})\} \end{aligned}$$

is in $\Pi^p_i$. Consequently, the set

$$F = \{\langle x, 1^n, k\rangle \mid (\exists\ \omega_s, \omega_{s-1}, \ldots, \omega_{s-k+2})[\langle x, 1^n, \omega_s, \omega_{s-1}, \ldots, \omega_{s-k+2}\rangle \in E]\}$$

is in $\Sigma^p_{i+1}$. Observe that if $x$ is a hard string for length $r'(\widehat{p}_1(n))$, then it is the case that $\langle x, 1^n, \omega_s, \omega_{s-1}, \ldots, \omega_j\rangle \in E$ if and only if $(x, \omega_s, \omega_{s-1}, \ldots, \omega_j)$ is a hard sequence for length $\widehat{p}_1(n)$. Similarly, if $x$ is a hard string for length $r'(\widehat{p}_1(n))$, then $\langle x, 1^n, k\rangle \in F$ if and only if there exists a hard sequence (starting with $(x, \ldots)$) *of length* $k$ for length $\widehat{p}_1(n)$.

It follows from those observations and the above proven claims that if $x$ is a hard string for length $r'(\widehat{p}_1(n))$, then the following algorithm will accept $\langle x, 1^n, y\rangle$ if and only if $y \in L^{\dagger}_{\Sigma^p_{i+2}}$. On input $\langle x, 1^n, y\rangle$ the algorithm proceeds as follows:

1. Using $F$ as an oracle, compute the largest $k$, call it $\widehat{k}$, such that $\langle x, 1^n, k\rangle \in F$.
2. Then, by making one oracle query, check the following: Do there exist strings $\omega_s, \omega_{s-1}, \ldots, \omega_{s-\widehat{k}+2}$ such that $\langle x, 1^n, \omega_s, \omega_{s-1}, \ldots, \omega_{s-\widehat{k}+2}\rangle \in E$ and $\langle x, 1^n, \omega_s, \omega_{s-1}, \ldots, \omega_{s-\widehat{k}+2}, y\rangle \in C$? Though we actually are allowed as many queries as we like (within our time bound), we note that it is not hard to see that this checking can be done by making one query to an appropriately chosen $\Sigma^p_{i+1}$ oracle.
3. Accept if and only if the final query returned the answer "yes."

Though the above algorithm queries two different $\Sigma^p_{i+1}$ oracles it is clearly a $\mathrm{P}^{\Sigma^p_{i+1}}$ algorithm, since $\Sigma^p_{i+1}$ is closed under disjoint union. Let $D$ be the set accepted by this algorithm. Define $r$ to be the polynomial such that $r(n) = r'(\widehat{p}_1(n))$ for all $n$. Note that due to the definitions of $r'$ and $\widehat{p}_1$, $r$ satisfies $r(n+1) > r(n) > 0$ for all $n$. This completes the proof of Lemma 4.3. $\square$

## 5. Conclusions.

We have proven a general downward translation of equality, Theorem 3.3, sufficient to yield, as a corollary, the following.

COROLLARY 5.1. *For each $m > 0$ and each $k > 1$,*

$$\mathrm{P}^{\Sigma^p_k}_{m\text{-tt}} = \mathrm{P}^{\Sigma^p_k}_{m+1\text{-tt}} \Rightarrow \mathrm{DIFF}_m(\Sigma^p_k) = \mathrm{coDIFF}_m(\Sigma^p_k).$$

The corollary follows immediately from Theorem 3.3, Proposition 2.2, and Observation 2.3. Corollary 5.1 itself has an interesting further consequence. From this corollary, it follows that for a number of previously missing cases (namely, when $m > 1$ and $k = 2$), the hypothesis $\mathrm{P}^{\Sigma^p_k}_{m\text{-tt}} = \mathrm{P}^{\Sigma^p_k}_{m+1\text{-tt}}$ implies that the polynomial hierarchy collapses to about one level lower in the boolean hierarchy over $\Sigma^p_{k+1}$ than could be concluded from previous papers. This is because we can, thanks to Corollary 5.1, when given $\mathrm{P}^{\Sigma^p_k}_{m\text{-tt}} = \mathrm{P}^{\Sigma^p_k}_{m+1\text{-tt}}$, invoke the powerful collapses of the polynomial hierarchy that are known to follow from $\mathrm{DIFF}_m(\Sigma^p_k) = \mathrm{coDIFF}_m(\Sigma^p_k)$. In particular, a long line of research started by Kadin [17] and Wagner [28, 29] over a decade ago has studied what collapses follow from $\mathrm{DIFF}_m(\Sigma^p_k) = \mathrm{coDIFF}_m(\Sigma^p_k)$. The strongest currently known connection was recently obtained, independently, by Hemaspaandra, Hemaspaandra, and Hempel [13, Theorem 5.1] and by Reith and Wagner [23, Corollary 27], namely: For all $m > 0$ and all $k > 0$, if $\mathrm{DIFF}_m(\Sigma^p_k) = \mathrm{coDIFF}_m(\Sigma^p_k)$, then $\mathrm{PH} = \mathrm{DIFF}_m(\Sigma^p_k) \bigoplus \mathrm{DIFF}_{m-1}(\Sigma^p_{k+1})$. Putting all the above together, one sees that, for all cases where $m > 1$ and $k > 1$, $\mathrm{P}^{\Sigma^p_k}_{m\text{-tt}} = \mathrm{P}^{\Sigma^p_k}_{m+1\text{-tt}}$ implies that the polynomial hierarchy collapses to $\mathrm{DIFF}_m(\Sigma^p_k) \bigoplus \mathrm{DIFF}_{m-1}(\Sigma^p_{k+1})$. This also yields that, for all cases where $m > 1$ and $k > 1$, $\mathrm{P}^{\Sigma^p_k[m]} = \mathrm{P}^{\Sigma^p_k[m+1]}$ implies that the polynomial hierarchy collapses to $\mathrm{DIFF}_{2^m-1}(\Sigma^p_k) \bigoplus \mathrm{DIFF}_{2^m-2}(\Sigma^p_{k+1})$. Of course, for the case $m = 1$, we already know [14, 4] that, for $k > 1$, if $\mathrm{P}^{\Sigma^p_k[1]} = \mathrm{P}^{\Sigma^p_k[2]}$ (equivalently, if $\mathrm{P}^{\Sigma^p_k}_{1\text{-tt}} = \mathrm{P}^{\Sigma^p_k}_{2\text{-tt}}$), then $\Sigma^p_k = \Pi^p_k = \mathrm{PH}$.

**Acknowledgments.** We thank the anonymous conference and journal referees for helpful comments and suggestions, and we thank the students of CSC486 at the University of Rochester for their proofreading help.

## REFERENCES

[1] E. Allender, *Limitations of the upward separation technique*, Math. Systems Theory, 24 (1991), pp. 53–67.

[2] E. Allender and C. Wilson, *Downward translations of equality*, Theoret. Comput. Sci., 75 (1990), pp. 335–346.

[3] R. Beigel, R. Chang, and M. Ogiwara, *A relationship between difference hierarchies and relativized polynomial hierarchies*, Math. Systems Theory, 26 (1993), pp. 293–310.

[4] H. Buhrman and L. Fortnow, *Two queries*, J. Comput. System Sci., 59 (1999), pp. 182–194.

[5] J. Cai, T. Gundermann, J. Hartmanis, L. Hemachandra, V. Sewelson, K. Wagner, and G. Wechsung, *The boolean hierarchy* I: *Structural properties*, SIAM J. Comput., 17 (1988), pp. 1232–1252.

[6] J. Cai, T. Gundermann, J. Hartmanis, L. Hemachandra, V. Sewelson, K. Wagner, and G. Wechsung, *The boolean hierarchy* II: *Applications*, SIAM J. Comput., 18 (1989), pp. 95–111.

[7] R. Chang, *Bounded queries, approximations, and the Boolean hierarchy*, Inform. and Comput., 169 (2001), pp. 129–159.

[8] R. Chang and J. Kadin, *The Boolean hierarchy and the polynomial hierarchy: A closer connection*, SIAM J. Comput., 25 (1996), pp. 340–354.

[9] J. Hartmanis, *On sparse sets in NP−P*, Inform. Process. Lett., 16 (1983), pp. 55–60.

[10] J. Hartmanis, N. Immerman, and V. Sewelson, *Sparse sets in NP−P: EXPTIME versus NEXPTIME*, Inform. and Control, 65 (1985), pp. 158–181.

[11] E. Hemaspaandra, L. Hemaspaandra, and H. Hempel, *An introduction to query order*, Bul. Eur. Assoc. Theor. Comput. Sci. EATCS, 63 (1997), pp. 93–107.

[12] E. Hemaspaandra, L. Hemaspaandra, and H. Hempel, *Query order in the polynomial hierarchy*, Journal of Universal Computer Science, 4 (1998), pp. 574–588.

[13] E. Hemaspaandra, L. Hemaspaandra, and H. Hempel, *What's up with downward collapse: Using the easy-hard technique to link boolean and polynomial hierarchy collapses*, SIGACT News, 29 (1998), pp. 10–22.

[14] E. Hemaspaandra, L. Hemaspaandra, and H. Hempel, *A downward collapse within the polynomial hierarchy*, SIAM J. Comput., 28 (1999), pp. 383–393.

[15] L. Hemaspaandra, H. Hempel, and G. Wechsung, *Query order*, SIAM J. Comput., 28 (1999), pp. 637–651.

[16] L. Hemaspaandra and S. Jha, *Defying upward and downward separation*, Inform. and Comput., 121 (1995), pp. 1–13.

[17] J. Kadin, *The polynomial time hierarchy collapses if the boolean hierarchy collapses*, SIAM J. Comput., 17 (1988), pp. 1263–1282. Erratum appears in the same journal, 20 (1991), p. 404.

[18] J. Köbler, U. Schöning, and K. Wagner, *The difference and truth-table hierarchies for NP*, RAIRO Theoret. Inform. and Appl., 21 (1987), pp. 419–435.

[19] R. Ladner, N. Lynch, and A. Selman, *A comparison of polynomial time reducibilities*, Theoret. Comput. Sci., 1 (1975), pp. 103–123.

[20] A. Meyer and L. Stockmeyer, *The equivalence problem for regular expressions with squaring requires exponential space*, in Proceedings of the 13th IEEE Symposium on Switching and Automata Theory, 1972, pp. 125–129.

[21] R. Rao, J. Rothe, and O. Watanabe, *Upward separation for FewP and related classes*, Inform. Process. Lett., 52 (1994), pp. 175–180.

[22] R. Rao, J. Rothe, and O. Watanabe, *Corrigendum to Upward separation for FewP and related classes*, Inform. Process. Lett., 74 (2000), pp. 89.

[23] S. Reith and K. Wagner, *On Boolean lowness and Boolean highness*, Theoret. Comput. Sci., 261 (2001), pp. 305–321.

[24] P. Rohatgi, *Saving queries with randomness*, J. Comput. System Sci., 50 (1995), pp. 476–492.

[25] V. Selivanov, *Two refinements of the polynomial hierarchy*, in Proceedings of the 11th Annual Symposium on Theoretical Aspects of Computer Science, Lecture Notes in Comput. Sci. 775, Springer-Verlag, Berlin, 1994, pp. 439–448.

[26] V. Selivanov, *Fine hierarchies and Boolean terms*, J. Symbolic Logic, 60 (1995), pp. 289–317.

[27] L. Stockmeyer, *The polynomial-time hierarchy*, Theoret. Comput. Sci., 3 (1976), pp. 1–22.

[28] K. Wagner, *Number-of-Query Hierarchies*, Tech. Report 158, Institut für Mathematik, Universität Augsburg, Augsburg, Germany, 1987.

[29] K. Wagner, *Number-of-Query Hierarchies*, Tech. Report 4, Institut für Informatik, Universität Würzburg, Würzburg, Germany, 1989.

[30] K. Wagner, *Bounded query classes*, SIAM J. Comput., 19 (1990), pp. 833–846.

[31] K. Wagner, *A note on parallel queries and the symmetric-difference hierarchy*, Inform. Process. Lett., 66 (1998), pp. 13–20.
[32] C. Wrathall, *Complete sets and the polynomial-time hierarchy*, Theoret. Comput. Sci., 3 (1976), pp. 23–33.

SIAM J. Comput.
Vol. 34, No. 6, pp. 1370–1379

# APPROXIMATING THE MINIMUM SPANNING TREE WEIGHT IN SUBLINEAR TIME[*]

BERNARD CHAZELLE[†], RONITT RUBINFELD[‡], AND LUCA TREVISAN[§]

**Abstract.** We present a probabilistic algorithm that, given a connected graph $G$ (represented by adjacency lists) of average degree $d$, with edge weights in the set $\{1, \ldots, w\}$, and given a parameter $0 < \varepsilon < 1/2$, estimates in time $O(dw\varepsilon^{-2} \log \frac{dw}{\varepsilon})$ the weight of the minimum spanning tree (MST) of $G$ with a relative error of at most $\varepsilon$. Note that the running time does *not* depend on the number of vertices in $G$. We also prove a nearly matching lower bound of $\Omega(dw\varepsilon^{-2})$ on the probe and time complexity of any approximation algorithm for MST weight.

The essential component of our algorithm is a procedure for estimating in time $O(d\varepsilon^{-2} \log \frac{d}{\varepsilon})$ the number of connected components of an unweighted graph to within an additive error of $\varepsilon n$. (This becomes $O(\varepsilon^{-2} \log \frac{1}{\varepsilon})$ for $d = O(1)$.) The time bound is shown to be tight up to within the $\log \frac{d}{\varepsilon}$ factor. Our connected-components algorithm picks $O(1/\varepsilon^2)$ vertices in the graph and then grows "local spanning trees" whose sizes are specified by a stochastic process. From the local information collected in this way, the algorithm is able to infer, with high confidence, an estimate of the number of connected components. We then show how estimates on the number of components in various subgraphs of $G$ can be used to estimate the weight of its MST.

**Key words.** minimum spanning tree, sublinear time algorithms, randomized algorithms, approximation algorithms

**AMS subject classifications.** 68W20, 68W25, 68R10

**DOI.** 10.1137/S0097539702403244

## 1. Introduction.

Traditionally, a linear time algorithm has been held as the gold standard of efficiency. In a wide variety of settings, however, large data sets have become increasingly common, and it is often desirable and sometimes necessary to find very fast algorithms which can assert nontrivial properties of the data in *sublinear* time.

One direction of research that has been suggested is that of property testing [16, 8], which relaxes the standard notion of a decision problem. Property testing algorithms distinguish between inputs that have a certain property and those that are far (in terms of Hamming distance or some other natural distance) from having the property. Sublinear and even constant time algorithms have been designed for testing various algebraic and combinatorial properties (see [15] for a survey). Property testing can be viewed as a natural type of approximation problem, and, in fact, many of the property testers have led to very fast, even constant time, approximation schemes for the associated problem (cf. [8, 5, 6, 1]). For example, one can approximate the

[*]Received by the editors February 27, 2002; accepted for publication (in revised form) November 9, 2004; published electronically July 26, 2005. A preliminary version of this work appeared in *Automata, Languages and Programming*, Lecture Notes in Comput. Sci. 2076, Springer-Verlag, Berlin, 2001, pp. 190–200.
http://www.siam.org/journals/sicomp/34-6/40324.html

[†]Department of Computer Science, Princeton University, Princeton, NJ, 08544 (chazelle@cs.princeton.edu). Part of this research was supported by NSF grant CCR-99817 and ARO grant DAAH04-96-1-0181.

[‡]Department of Electrical Engineering and Computer Science, MIT, Cambridge, MA, 02139 (ronitt@csail.mit.edu). Part of this work was done while the author was at the NEC Research Institute.

[§]Computer Science Division, University of California–Berkeley, Berkeley, CA 94720 (luca@eecs.berkeley.edu).

value of a maximum cut in a dense graph in time $2^{O(\varepsilon^{-3}\log 1/\varepsilon)}$, with relative error at most $\varepsilon$, by looking at only $O(\varepsilon^{-7}\log 1/\varepsilon)$ locations in the adjacency matrix [8]. Other sublinear time approximation schemes have been applied to dense instances of graph bisection, general partitioning problems, quadratic assignment, minimum linear arrangement, and maximum acyclic subgraph and constraint satisfaction [8, 5], as well as clustering [1, 14]. Note that typically such schemes approximate the value of the optimal solution, for example, the size of a maxcut, without computing the structure that achieves it, i.e., the actual cut. Sometimes, however, a solution can also be constructed in linear or near-linear time.

In this paper, we consider the problem of finding the weight of the minimum spanning tree (MST) of a graph. Finding the MST of a graph has a long, distinguished history [3, 10, 12]. Currently the best known deterministic algorithm of Chazelle [2] runs in $O(m\alpha(m,n))$ time, where $n$ (resp., $m$) is the number of vertices (resp., edges) and $\alpha$ is inverse-Ackermann. The randomized algorithm of Karger, Klein, and Tarjan [11] runs in linear expected time (see also [4, 13] for alternative models).

In this paper, we show that there are conditions under which it is possible to approximate the weight of the MST of a connected graph in time sublinear in the number of edges. We give an algorithm which approximates the MST of a graph $G$ to within a multiplicative factor of $1+\varepsilon$ and runs in time $O(dw\varepsilon^{-2}\log\frac{dw}{\varepsilon})$ for any $G$ with average degree $d$ and edge weights in the set $\{1,\ldots,w\}$. The algorithm requires no prior information about the graph besides $w$ and $n$; in particular, the average degree is assumed to be unknown. The relative error $\varepsilon$ $(0<\varepsilon<1/2)$ is specified as an input parameter. Note that if $d$ and $\varepsilon$ are constant and the ratios of the edge weights are bounded, then the algorithm runs in constant time. We also extend our algorithm to the case where $G$ has nonintegral weights in the range $[1,w]$, achieving a comparable running time with a somewhat worse dependence on $\varepsilon$.

Our algorithm considers several auxiliary graphs: If $G$ is the weighted graph, let us denote by $G^{(i)}$ the subgraph of $G$ that contains only edges of weight at most $i$. We estimate the number of connected components in each $G^{(i)}$. To do so, we sample uniformly at random $O(1/\varepsilon^2)$ vertices in $G^{(i)}$ and then estimate the size of the component that contains each sampled vertex by constructing "local trees" of some appropriate size defined by a random process. Based on information about these local trees, we can in turn produce a good approximation for the weight of the MST of $G$. Our algorithm for estimating the number of connected components in a graph runs in time $O(d\varepsilon^{-2}\log\frac{d}{\varepsilon})$—or $O(\varepsilon^{-2}\log\frac{1}{\varepsilon})$ for $d=O(1)$—and produces an estimate that is within an additive error of $\varepsilon n$ of the true count. The method is based on a similar principle as the property tester for graph connectivity given by Goldreich and Ron [9].

We give a lower bound of $\Omega(dw/\varepsilon^2)$ on the time complexity of any algorithm which approximates the MST weight. In order to prove the lower bound, we give two distributions on weighted graphs, where the support set of one distribution contains graphs with MST weight at least $1+\varepsilon$ times the MST weight of the graphs in the support of the other distribution. We show that any algorithm that reads $o(dw/\varepsilon^2)$ weights from the input graph is unlikely to distinguish between graphs from the two distributions. We also prove a lower bound of $\Omega(d/\varepsilon^2)$ on the running time of any approximation algorithm for counting connected components. The above lower bounds apply to the class of graphs which may contain self-loops and multiple edges.

## 2. Estimating the number of connected components.

We begin with the problem of estimating the number of components in an arbitrary graph $G$. For no-

```
approx-number-connected-components($G, \varepsilon, W, d^*$)
  uniformly choose $r = O(1/\varepsilon^2)$ vertices $u_1, \ldots, u_r$
  for each vertex $u_i$,
    set $\beta_i = 0$
    take the first step of a BFS from $u_i$
    (*) flip a coin
    if (heads) & (# vertices visited in BFS $< W$)
              & (no visited vertex has degree $> d^*$)
      then {resume BFS to double number of visited edges
            if this allows BFS to complete
              then {if $m_{u_i} = 0$ set $\beta_i = 2$
                    else set $\beta_i = d_{u_i} 2^{\#\text{coin flips}} / \#\text{edges}$ visited in BFS }
              else go to (*) }
  output $\hat{c} = \frac{n}{2r} \sum_{i=1}^{r} \beta_i$
```

FIG. 1. *Estimating the number of connected components; see main text for precise definition of BFS.*

tational convenience, we may assume that $d \geq 1$: This can always be achieved by implicitly adding a fictitious self-loop to each vertex, which does not change the number of connected components. We present an algorithm that gives an additive estimate of the number of components in $G$ to within $\varepsilon n$ in $O(d\varepsilon^{-2} \log \frac{d}{\varepsilon})$ time for any $0 < \varepsilon < 1/2$. We later show how to use the ideas from our algorithm to aid in estimating the weight of the MST of a graph. We use the following notation: Given a vertex $u$, $d_u$ is the number of edges incident upon it (including self-loops), and $m_u$ is the number of edges in $u$'s component in $G$. Finally, $c$ denotes the number of connected components. Our algorithm is built around the following simple observation.

FACT 1. *Given a graph with vertex set $V$, for every connected component $I \subseteq V$, $\sum_{u \in I} \frac{1}{2} d_u / m_u = 1$ and $\sum_{u \in V} \frac{1}{2} d_u / m_u = c$.*

To handle isolated vertices, we must make the convention that $d_u/m_u = 2$ if $m_u = 0$. Our strategy is to estimate $c$ by approximating each $d_u/m_u$. Computing them directly could take linear time, so we construct an estimator of the quantity $d_u/m_u$ that has the same expected value. We approximate the number of connected components via the algorithm given in Figure 1. The parameter $W$ is a threshold value which is set to $4/\varepsilon$ for counting connected components and somewhat higher for MST weight estimation. We also use an estimate $d^*$ of the average degree $d$, which we compute separately in $O(d/\varepsilon)$ expected time (see Lemma 4). This approximation ensures that $d^* = O(d/\varepsilon)$ and that at most $\varepsilon n/4$ vertices have degree higher than $d^*$.

In the algorithm, doubling the number of edges does not include duplicate visits to the same edges; in other words, at each phase the number of new edges visited is supposed to match the number of distinct edges already visited. In our terminology, the first step of the BFS (shorthand for breadth first search) involves the visit of the single vertex $u_i$ and all its $d_{u_i}$ incident edges. That is, unless $d_{u_i} > d^*$, in which case we abort the BFS.

We now bound the expectation and variance of the estimator $\beta_i$ for a fixed $i$. If the BFS from $u_i$ completes, the number of coin flips associated with it is $\lceil \log(m_{u_i}/d_{u_i}) \rceil$, and the number of distinct edges visited is $m_{u_i}$. Let $S$ denote the set of vertices that lie in components with fewer than $W$ vertices all of which are of degree at most $d^*$. If $u_i \notin S$, then $\beta_i = 0$; otherwise, it is $2^{\lceil \log(m_{u_i}/d_{u_i}) \rceil} d_{u_i}/m_{u_i}$ with probability

$2^{-\lceil \log(m_{u_i}/d_{u_i})\rceil}$ (and 2 if $m_{u_i} = 0$) and 0 otherwise. Since $\beta_i \leq 2$, the variance of $\beta_i$ is

$$\mathbf{var}\,\beta_i \leq \mathbf{E}\,\beta_i^2 \leq 2\mathbf{E}\,\beta_i = \frac{2}{n}\sum_{u\in S}\frac{d_u}{m_u} \leq \frac{4c}{n}\,.$$

Then the variance of $\hat{c}$ is bounded by

$$\mathbf{var}\,\hat{c} = \mathbf{var}\left(\frac{n}{2r}\sum_i \beta_i\right) = \frac{n^2}{4r^2}\cdot r\cdot \mathbf{var}\,\beta_i \leq \frac{nc}{r}. \tag{1}$$

By our choice of $W = 4/\varepsilon$ and $d^*$, there are at most $\varepsilon n/2$ components with vertices not in $S$, and so

$$c - \frac{\varepsilon n}{2} \leq \mathbf{E}\,\hat{c} \leq c\,. \tag{2}$$

Furthermore, by Chebyshev,

$$\text{Prob}[\,|\hat{c} - \mathbf{E}\,\hat{c}| > \varepsilon n/2\,] < \frac{\mathbf{var}\,\hat{c}}{(\varepsilon n/2)^2} \leq \frac{4c}{\varepsilon^2 rn}\,. \tag{3}$$

Choosing $r = O(1/\varepsilon^2)$ ensures that, with constant probability arbitrarily close to 1, our estimate $\hat{c}$ of the number of connected components deviates from the actual value by at most $\varepsilon n$.

The expected number of edges visited in a given iteration of the "for loop" is $O(d_{u_i}\log M)$, where $M$ is the maximum number of edges visited, which is at most $Wd^* = O(d/\varepsilon^2)$. Therefore, the expected running time of the entire algorithm is

$$\frac{O(r)}{n}\sum_{u\in V} d_u \log(Wd^*) = O(dr\log(Wd^*)) = O\left(d\varepsilon^{-2}\log\frac{d}{\varepsilon}\right)\,, \tag{4}$$

not counting the $O(d/\varepsilon)$ time needed for computing $d^*$.

As stated, the algorithm's running time is randomized. If $d$ is known, however, we can get a deterministic running time bound by stopping the algorithm after $Cd\varepsilon^{-2}\log\frac{d}{\varepsilon}$ steps and outputting 0 if the algorithm has not yet terminated. This event occurs with probability at most $O(1/C)$, which is a negligible addition to the error probability. Thus we have the following theorem.

THEOREM 2. *Let $c$ be the number of components in a graph with $n$ vertices. Then Algorithm* approx-number-connected-components *runs in time $O(d\varepsilon^{-2}\log\frac{d}{\varepsilon})$ and with probability at least $3/4$ outputs $\hat{c}$ such that $|c - \hat{c}| \leq \varepsilon n$.*

*Finetuning the algorithm.* If we proceed in two stages, first estimating $c$ within a constant factor, and then in a second pass using this value to optimize the size of the sample, we can lower the running time to $O((\varepsilon + c/n)d\varepsilon^{-2}\log\frac{d}{\varepsilon})$. This is a substantial improvement for small values of $c$. First, run the algorithm for $r = O(1/\varepsilon)$. By Chebyshev and (1, 2),

$$\text{Prob}\left[\,|\hat{c} - \mathbf{E}\,\hat{c}| > \frac{\mathbf{E}\,\hat{c} + \varepsilon n}{2}\,\right] < \frac{4nc}{r(c+\varepsilon n/2)^2} \leq \frac{4n}{r(c+\varepsilon n/2)},$$

which is arbitrarily small for $r\varepsilon$ large enough. Next, we use this approximation $\hat{c}$ to "improve" the value of $r$. We set $r = A/\varepsilon + A\hat{c}/(\varepsilon^2 n)$ for some large enough constant

$A$ and we run the algorithm again, with the effect of producing a second estimate $c^*$. By (2, 3),

$$\text{Prob}[\,|c^* - \mathbf{E}\,c^*| > \varepsilon n/2\,] < \frac{4c}{\varepsilon^2 rn} \leq \frac{8c}{A\varepsilon n + A\mathbf{E}\,\hat{c}} \leq \frac{8}{A},$$

and so, with overwhelming probability, our second estimate $c^*$ of the number of connected components deviates from $c$ by at most $\varepsilon n$. So we have the following theorem.

THEOREM 3. *Let $c$ be the number of components in a graph with $n$ vertices. Then there is an algorithm that runs in time $O(d\varepsilon^{-2} \log \frac{d}{\varepsilon})$ and with probability at least $3/4$ outputs $\hat{c}$ such that $|c - \hat{c}| \leq \varepsilon n$.*

*Approximating the degree.* We show how to compute the desired estimate $d^*$ of the average degree $d$. Pick $C/\varepsilon$ vertices of $G$ at random, for some large constant $C$, and set $d^*$ to be the maximum degree among them. To find the degree of any one of them takes $O(d)$ time on average, and so the expected running time is $O(d/\varepsilon)$. Imagine the vertex degrees sorted in nonincreasing order, and let $\rho$ be the rank of $d^*$. With high probability, $\rho = \Theta(\varepsilon n)$. To see why, we easily bound the probability that $\rho$ exceeds $\varepsilon n$ by $(1-\varepsilon)^{C/\varepsilon} \leq e^{-C}$. On the other hand, observe that the probability that $\rho > \varepsilon n/C^2$ is at least $(1-\varepsilon/C^2)^{C/\varepsilon} \geq e^{-2/C} > 1 - 2/C$.

LEMMA 4. *In $O(d/\varepsilon)$ expected time, we can compute a vertex degree $d^*$ that, with high probability, is the $k$th largest vertex degree for some $k = \Theta(\varepsilon n)$.*

Note that $k = \Omega(\varepsilon n)$ alone implies that $d^* = O(d/\varepsilon)$, and so, if we scale $\varepsilon$ by the proper constant, we can ensure that at most $\varepsilon n/4$ vertices have degree higher than $d^*$, and thus conform to the requirements of approx-number-connected-components.

## 3. Approximating the weight of an MST.

In this section we present an algorithm for approximating the value of the MST in bounded weight graphs. We are given a connected graph $G$ with average degree $d$ and with each edge assigned an integer weight between 1 and $w$. We assume that $G$ is represented by adjacency lists or, for that matter, any representation that allows one to access all edges incident to a given vertex in $O(d)$ time. We show how to approximate the weight of the MST of $G$ with a relative error of at most $\varepsilon$.

In section 3.1 we give a new way to characterize the weight of the MST in terms of the number of connected components in subgraphs of $G$. In section 3.2 we give the main algorithm and its analysis. Finally, section 3.3 addresses how to extend the algorithm to the case where $G$ has nonintegral weights.

### 3.1. MST weight and connected components.

We reduce the computation of the MST weight to counting connected components in various subgraphs of $G$. To motivate the new characterization, consider the special case when $G$ has only edges of weight 1 or 2 (i.e., $w = 2$). Let $G^{(1)}$ be the subgraph of $G$ consisting precisely of the edges of weight 1, and let $n_1$ be its number of connected components. Then, any MST in $G$ must contain exactly $n_1 - 1$ edges of weight 2, with all the others being of weight 1. Thus, the weight of the MST is exactly $n - 2 + n_1$. We easily generalize this derivation to any $w$.

For each $0 \leq \ell \leq w$, let $G^{(\ell)}$ denote the subgraph of $G$ consisting of all the edges of weight at most $\ell$. Define $c^{(\ell)}$ to be the number of connected components in $G^{(\ell)}$ (with $c^{(0)}$ defined to be $n$). By our assumption on the weights, $c^{(w)} = 1$. Let $M(G)$ be the weight of the MST of $G$. Using the above quantities, we give an alternate way of computing the value of $M(G)$ in the following claim.

```
approx-MST-weight($G, \varepsilon$)
  For $i = 1, \dots, w-1$
    $\hat{c}^{(i)} =$ approx-number-connected-components($G^{(i)}, \varepsilon, 4w/\varepsilon, d^*$)
  output $\hat{v} = n - w + \sum_{i=1}^{w-1} \hat{c}^{(i)}$
```

FIG. 2. *Approximating the weight of the MST.*

CLAIM 5. *For integer $w \geq 2$,*

$$M(G) = n - w + \sum_{i=1}^{w-1} c^{(i)} .$$

*Proof.* Let $\alpha_i$ be the number of edges of weight $i$ in an MST of $G$. (Note that $\alpha_i$ is independent of which MST we choose [7].) Observe that for all $0 \leq \ell \leq w-1$, $\sum_{i>\ell} \alpha_i = c^{(\ell)} - 1$; therefore

$$M(G) = \sum_{i=1}^{w} i\alpha_i = \sum_{\ell=0}^{w-1} \sum_{i=\ell+1}^{w} \alpha_i = -w + \sum_{\ell=0}^{w-1} c^{(\ell)} = n - w + \sum_{i=1}^{w-1} c^{(i)}. \qquad \square$$

Thus, computing the number of connected components allows us to compute the weight of the MST of $G$.

**3.2. The main algorithm.** Our algorithm approximates the value of the MST by estimating each of the $c^{(\ell)}$'s. The algorithm is given in Figure 2. Note that we do not set $W = 4/\varepsilon$ in the call to the connected-components algorithm. For the same reason (to be explained below) we need a different estimate of the degree $d^*$. We use Lemma 4 just once to compute, in $O(dw/\varepsilon)$ time, an estimate $d^* = O(dw/\varepsilon)$ such that at most $\varepsilon n/4w$ vertices have degree higher than $d^*$.

In the following, we assume that $w/n < 1/2$, since otherwise we might as well compute the MST explicitly, which can be done in $O(dn)$ time with high probability [11].

THEOREM 6. *Let $w/n < 1/2$. Let $v$ be the weight of the MST of $G$. Algorithm* approx-MST-weight *runs in time $O(dw\varepsilon^{-2} \log \frac{dw}{\varepsilon})$ and outputs a value $\hat{v}$ that, with probability at least $3/4$, differs from $v$ by at most $\varepsilon v$.*

*Proof.* Let $c = \sum_{i=1}^{w-1} c^{(i)}$. Repeating the previous analysis, we find that (1), (2) become

$$c^{(i)} - \frac{\varepsilon n}{2w} \leq \mathbf{E}\, \hat{c}^{(i)} \leq c^{(i)} \qquad \text{and} \qquad \mathbf{var}\, \hat{c}^{(i)} \leq \frac{nc^{(i)}}{r} .$$

By summing over $i$, it follows that $c - \varepsilon n/2 \leq \mathbf{E}\, \hat{c} \leq c$ and $\mathbf{var}\, \hat{c} \leq nc/r$, where $\hat{c} = \sum_{i=1}^{w-1} \hat{c}^{(i)}$. Choosing $r\varepsilon^2$ large enough, by Chebyshev we have

$$\text{Prob}[\, |\hat{c} - \mathbf{E}\, \hat{c}| > (n - w + c)\varepsilon/3\,] < \frac{9nc}{r\varepsilon^2 (n - w + c)^2},$$

which is arbitrarily small. It follows that, with high probability, the error on the estimate satisfies

$$|v - \hat{v}| = |c - \hat{c}| \leq \frac{\varepsilon n}{2} + \frac{\varepsilon(n - w + c)}{3} \leq \varepsilon v.$$

Since, by (4), the expected running time of each call to approx-number-connected-components is $O(dr\log(Wd^*))$, the total expected running time is $O(dw\varepsilon^{-2}\log\frac{dw}{\varepsilon})$. As before, if we know $d$, then the running time can be made deterministic by stopping execution of the algorithm after $Cdw\varepsilon^{-2}\log\frac{dw}{\varepsilon}$ steps for some appropriately chosen constant $C$. $\square$

**3.3. Nonintegral weights.** Suppose the weights of $G$ are all in the range $[1, w]$, but are not necessarily integral. To extend the algorithm to this case, one can multiply all the weights by $1/\varepsilon$ and round each weight to the nearest integer. Then one can run the above algorithm with error parameter $\varepsilon/2$ and with a new range of weights $[1, \lceil w/\varepsilon\rceil]$ to get a value $v$. Finally, output $\varepsilon v$. The relative error introduced by the rounding is at most $\varepsilon/2$ per edge in the MST and hence $\varepsilon/2$ for the whole MST, which gives a total relative error of at most $\varepsilon$. The running time of the above algorithm is $O(dw\varepsilon^{-3}\log\frac{w}{\varepsilon})$.

## 4. Lower bounds.

We prove that our algorithms for estimating the MST weight and counting connected components are essentially optimal. Our lower bounds apply to graphs that may contain self-loops and multiple edges.

THEOREM 7. *Any probabilistic algorithm for approximating, with relative error $\varepsilon$, the MST weight of a connected graph with average degree $d$ and weights in $\{1, \dots, w\}$ requires $\Omega(dw\varepsilon^{-2})$ edge weight lookups on average. It is assumed that $w > 1$ and $C\sqrt{w/n} < \varepsilon < 1/2$, for some large enough constant $C$.*

We can obviously assume that $w > 1$; otherwise the MST weight is always $n-1$ and no work is required. The lower bound on $\varepsilon$ might seem restrictive, but it is not at all. Indeed, by monotonicity on $\varepsilon$, the theorem implies a lower bound of $\Omega(dw(C\sqrt{w/n})^{-2})$ for any $\varepsilon \le C\sqrt{w/n}$. But this is $\Omega(dn)$, which we know is tight. Therefore, the case $C\sqrt{w/n} < \varepsilon$ is the only one that deserves attention.

THEOREM 8. *Given a graph with $n$ vertices and average degree $d$, any probabilistic algorithm for approximating the number of connected components with an additive error of $\varepsilon n$ requires $\Omega(d\varepsilon^{-2})$ edge lookups on average. It is assumed that $C/\sqrt{n} < \varepsilon < 1/2$, for some large enough constant $C$.*

Again, note that the lower bound on $\varepsilon$ is nonrestrictive since we can always solve the problem exactly in $O(dn)$ time. (For technical reasons, we allow graphs to have self-loops.)

Both proofs revolve around the difficulty of distinguishing between two nearby distributions. For any $0 < q \le 1/2$ and $s = 0, 1$, let $\mathcal{D}_q^s$ denote the distribution induced by setting a 0/1 random variable to 1 with probability $q_s = q(1 + (-1)^s\varepsilon)$. We define a distribution $\mathcal{D}$ on $n$-bit strings as follows: (1) pick $s = 1$ with probability 1/2 (and 0 else); (2) then draw a random string from $\{0,1\}^n$ (by choosing each $b_i$ from $\mathcal{D}_q^s$ independently). Consider a probabilistic algorithm that, given access to such a random bit string, outputs an estimate on the value of $s$. How well can it do?

LEMMA 9. *Any probabilistic algorithm that can guess the value of $s$ with a probability of error below $1/4$ requires $\Omega(\varepsilon^{-2}/q)$ bit lookups on average.*

*Proof.* By Yao's minimax principle, we may assume that the algorithm is deterministic and that the input is distributed according to $\mathcal{D}$. It is intuitively obvious that any algorithm might as well scan $b_1 b_2 \cdots$ until it decides it has seen enough to produce an estimate of $s$. In other words, there is no need to be adaptive in the choice of bit indices to probe (but the running time itself can be adaptive). To see why is easy. An algorithm can be modeled as a binary tree with a bit index at each node and a 0/1 label at each edge. An adaptive algorithm may have an arbitrary set of bit indices at the nodes, although we can assume that the same index does not appear twice along any path. Each leaf is naturally associated with a probability, which is that of a random input from $\mathcal{D}$ following the path to that leaf. The performance of the algorithm is entirely determined by these probabilities and the corresponding estimates of $s$. Because of the independence of the random $b_i$'s, we can relabel the tree so that each path is a prefix of the same sequence of bit probes $b_1 b_2 \cdots$. This oblivious algorithm has the same performance as the adaptive one.

We can go one step further and assume that the running time is the same for all inputs. Let $t^*$ be the expected number of probes, and let $0 < \alpha < 1$ be a small constant. With probability at most $\alpha$, a random input takes time $\geq t \overset{\text{def}}{=} t^*/\alpha$. Suppose that the prefix of bits examined by the algorithm is $b_1 \cdots b_u$. If $u < t$, simply go on probing $b_{u+1} \cdots b_t$ without changing the outcome. If $u > t$, then stop at $b_t$ and output $s = 1$. Thus, by adding $\alpha$ to the probability of error, we can assume that the algorithm consists of looking up $b_1 \cdots b_t$ regardless of the input string.

Let $p_s(b_1 \cdots b_t)$ be the probability that a random $t$-bit string chosen from $\mathcal{D}_q^s$ is equal to $b_1 \cdots b_t$. The probability of error satisfies

$$p_{\text{err}} \geq \frac{1}{2} \sum_{b_1 \cdots b_t} \min_s p_s(b_1 \cdots b_t).$$

Obviously, $p_s(b_1 \cdots b_t)$ depends only on the number of ones in the string, so if $p_s(k)$ denotes the probability that $b_1 + \cdots + b_t = k$, then

$$p_{\text{err}} \geq \frac{1}{2} \sum_{k=0}^{t} \min_s p_s(k). \tag{5}$$

By the normal approximation of the binomial distribution,

$$p_s(k) \to \frac{1}{\sqrt{2\pi t q_s(1-q_s)}}\, e^{-\frac{(k-tq_s)^2}{2tq_s(1-q_s)}}$$

as $t \to \infty$. This shows that $p_s(k) = \Omega(1/\sqrt{qt}\,)$ over an interval $I_s$ of length $\Omega(\sqrt{qt}\,)$ centered at $tq_s$. If $qt\varepsilon^2$ is smaller than a suitable constant $\gamma_0$, then $|tq_0 - tq_1|$ is small enough that $I_0 \cap I_1$ is itself an interval of length $\Omega(\sqrt{qt}\,)$; therefore $p_{\text{err}} = \Omega(1)$. This shows that if the algorithm runs in expected time $\gamma_0 \varepsilon^{-2}/q$, for some constant $\gamma_0 > 0$ small enough, then it will fail with probability at least some absolute constant. By setting $\alpha$ small enough, we can make that constant larger than $2\alpha$. This means that, prior to uniformizing the running time, the algorithm must still fail with probability $\alpha$.

Note that by choosing $\gamma_0$ small enough, we can always assume that $\alpha > 1/4$. Indeed, suppose by contradiction that even for an extremely small $\gamma_1$, there is an algorithm that runs in time at most $\gamma_1 \varepsilon^{-2}/q$ and fails with probability $\leq 1/4$. Then run the algorithm many times and take a majority vote. In this way we can bring the failure probability below $\alpha$ for a suitable $\gamma_1 = \gamma_1(\alpha, \gamma_0) < \gamma_0$ and therefore reach a contradiction. This means that an expected time lower than $\varepsilon^{-2}/q$ by a large enough constant factor causes a probability of error at least $1/4$. $\square$

*Proof of Theorem* 8. Consider the graph $G$ consisting of a simple cycle of $n$ vertices $v_1, \ldots, v_n$. Pick $s \in \{0, 1\}$ at random and take a random $n$-bit string $b_1 \cdots b_n$ with bits drawn independently from $\mathcal{D}_{1/2}^s$. Next, remove from $G$ any edge $(v_i, v_{i+1 \bmod n})$ if $b_i = 0$. Because $\varepsilon > C/\sqrt{n}$, the standard deviation of the number of components, which is $\Theta(\sqrt{n})$, is sufficiently smaller than $\varepsilon n$ so that with overwhelming probability any two graphs derived from $\mathcal{D}^0_{1/2}$ and $\mathcal{D}^1_{1/2}$ differ by more than $\varepsilon n/2$ in their numbers of connected components. That means that any probabilistic algorithm that estimates the number of connected components with an additive error of $\varepsilon n/2$ can be used to identify the correct $s$. By Lemma 9, this requires $\Omega(\varepsilon^{-2})$ edge probes into $G$ on average. Replacing $\varepsilon$ by $2\varepsilon$ proves Theorem 8 for graphs of average degree about 1.

For values of $d$ smaller than one, we may simply build a graph of the previous type on a fraction $d$ of the $n$ vertices and leave the others isolated. The same lower bound still holds as long as $d\varepsilon^2 n$ is bigger than a suitable constant. If $d > 1$, then we may simply add $d \pm O(1)$ self-loops to each vertex in order to bring the average degree up to $d$. Each linked list thus consists of two "cycle" pointers and about $d$ "loop" pointers. If we place the cycle pointers at random among the loop pointers, then it takes $\Omega(d)$ probes on average to hit a cycle pointer. If we single out the probes involving cycle pointers, it is not hard to argue that the probes involving cycle pointers are alone sufficient to solve the connected-components problem on the graph deprived of its loops: One expects at most $O(T/d)$ such probes, and therefore $T = \Omega(d\varepsilon^{-2})$. $\square$

*Proof of Theorem* 7. The input graph $G$ is a simple path of $n$ vertices. Pick $s \in \{0, 1\}$ at random and take a random $(n-1)$-bit string $b_1 \cdots b_{n-1}$ with bits drawn independently from $\mathcal{D}^s_q$, where $q = 1/w$. Assign weight $w$ (resp., 1) to the $i$th edge along the path if $b_i = 1$ (resp., 0). The MST of $G$ has weight $n - 1 + (w-1)\sum b_i$, and so its expectation is $\Theta(n)$. Also, note that the difference $\Delta$ in expectations between drawing from $\mathcal{D}^0_q$ or $\mathcal{D}^1_q$ is $\Theta(\varepsilon n)$.

Because $\varepsilon > C\sqrt{w/n}$, the standard deviation of the MST weight, which is $\Theta(\sqrt{nw})$, is sufficiently smaller than $\Delta$ that with overwhelming probability any two graphs derived from $\mathcal{D}^0_q$ and $\mathcal{D}^1_q$ differ by more than $\Delta/2$ in MST weight. Therefore, any probabilistic algorithm that estimates the weight with a relative error of $\varepsilon/D$, for some large enough constant $D$, can be used to identify the correct $s$. By Lemma 9, this means that $\Omega(w\varepsilon^{-2})$ probes into $G$ are required on average.

In this construction, $d = 2 - 2/n$ (the smallest possible value for a connected graph). For higher values of $d$, we join each vertex in the cycle to about $d-2$ others (say, at distance $> 2$ to avoid introducing multiple edges) to drive the degree up to $d$. Also, as usual, we randomize the ordering in each linked list. Assign weight $w+1$ to the new edges. (Allowing the maximum weight to be $w+1$ instead of $w$ has no influence on the lower bound for which we are aiming.) Clearly none of the new edges are used in the MST, so the problem is the same as before, except that we now have to find our way amidst $d-2$ spurious edges, which takes the complexity to $\Omega(dw\varepsilon^{-2})$. $\square$

## 5. Open questions.

Our algorithm for the case of nonintegral weights requires extra time. Is this necessary? Can the ideas in this paper be extended to finding maximum weighted independent sets in general matroids? There are now a small number of examples of approximation problems that can be solved in sublinear time; what other problems lend themselves to sublinear approximation schemes? More generally, it would be interesting to gain a more global understanding of what can and cannot be approximated in sublinear time.

**Acknowledgments.** We wish to thank David Silver for his helpful comments about the paper and for implementing the MST algorithm and showing that it works well in practice. We also thank the referees for many useful suggestions.

## REFERENCES

[1] N. Alon, S. Dar, M. Parnas, and D. Ron, *Testing of clustering,* SIAM J. Discrete Math., 16 (2003), pp. 393–417.

[2] B. Chazelle, *A minimum spanning tree algorithm with inverse-Ackermann type complexity*, J. ACM, 47 (2000), pp. 1028–1047.

[3] B. Chazelle, *The Discrepancy Method: Randomness and Complexity*, Cambridge University Press, Cambridge, UK, 2000.

[4] M. L. Fredman and D. E. Willard, *Trans-dichotomous algorithms for minimum spanning trees and shortest paths*, J. Comput. System Sci., 48 (1994), pp. 533–551.

[5] A. Frieze and R. Kannan, *Quick approximation to matrices and applications,* Combinatorica, 19 (1999), pp. 175–220.

[6] A. Frieze, R. Kannan, and S. Vempala, *Fast Monte-Carlo algorithms for finding low-rank approximations*, J. ACM, 51 (2004), pp. 1025–1041.

[7] D. Gale, *Optimal assignments in an ordered set: An application of matroid theory*, J. Combinatorial Theory, 4 (1968), pp. 176–180.

[8] O. Goldreich, S. Goldwasser, and D. Ron, *Property testing and its connection to learning and approximation*, J. ACM, 45 (1998), pp. 653–750.

[9] O. Goldreich and D. Ron, *Property testing in bounded degree graphs*, Algorithmica, 32 (2002), pp. 302–343.

[10] R. L. Graham and P. Hell, *On the history of the minimum spanning tree problem*, Ann. Hist. Comput., 7 (1985), pp. 43–57.

[11] D. R. Karger, P. N. Klein, and R. E. Tarjan, *A randomized linear-time algorithm to find minimum spanning trees,* J. ACM, 42 (1995), pp. 321–328.

[12] J. Nešetřil, *A few remarks on the history of MST-problem*, Arch. Math. (Brno), 33 (1997), pp. 15–22.

[13] S. Pettie and V. Ramachandran, *An optimal minimum spanning tree algorithm*, in Automata, Languages and Programming (Geneva, 2000), Lecture Notes in Comput. Sci. 1853, Springer-Verlag, Berlin, 2000, pp. 49–60.

[14] N. Mishra, D. Oblinger, and L. Pitt, *Sublinear time approximate clustering*, in Proceedings of the Twelfth Annual ACM-SIAM Symposium on Discrete Algorithms, ACM, New York, SIAM, Philadelphia, 2001, pp. 439–447.

[15] D. Ron, *Property testing,* in Handbook of Randomized Computing, Vol. II, S. Rajasekaran, P. M. Pardalos, J. H. Reif, and J. D. P. Rolim, eds., Kluwer Academic Publishers, Dordrecht, The Netherlands, 2001, pp. 597–649.

[16] R. Rubinfeld and M. Sudan, *Robust characterizations of polynomials with applications to program testing,* SIAM J. Comput., 25 (1996), pp. 252–271.

SIAM J. COMPUT. © 2005 Society for Industrial and Applied Mathematics
Vol. 34, No. 6, pp. 1380–1397

# BINARY SPACE PARTITIONS OF ORTHOGONAL SUBDIVISIONS*

JOHN HERSHBERGER†, SUBHASH SURI‡, AND CSABA D. TÓTH§

**Abstract.** We consider the problem of constructing binary space partitions (BSPs) for orthogonal subdivisions (space-filling packings of boxes) in $d$-space. We show that a subdivision with $n$ boxes can be refined into a BSP of size $O(n^{(d+1)/3})$ for all $d \geq 3$ and that such a partition can be computed in time $O(K \log n)$, where $K$ is the size of the BSP produced. Our upper bound on the BSP size is tight for 3-dimensional subdivisions; in higher dimensions, this is the first nontrivial result for general full-dimensional boxes. We also present a lower bound construction for a subdivision of $n$ boxes in $d$-space for which every axis-aligned BSP has $\Omega(n^{\beta(d)})$ size, where $\beta(d)$ converges to $(1+\sqrt{5})/2$ as $d \to \infty$.

**Key words.** binary space partitions, tilings

**AMS subject classifications.** 52C45, 68U05, 05D05

**DOI.** 10.1137/S0097539704445706

## 1. Introduction.

Many algorithms in computational geometry, robotics, and computer graphics decompose a set of objects and the ambient space for efficient processing of certain queries. A binary space partition (BSP) is a popular scheme for constructing such decompositions. Given an open convex region of space containing a set of pairwise disjoint objects $S$, a BSP partitions the region and objects with a cutting hyperplane, then recursively partitions the two resulting subproblems. The process stops when each open partition region intersects at most one object of $S$. See Figure 1 in section 2 for an example in two dimensions. In the ideal case, the number of regions in the final BSP would be at most $n$, the number of input objects. In general, however, the recursive partitioning may fracture input objects many times, and the size of the partition can be much larger than $n$.

BSPs were introduced in the computer graphics community [10, 16] to solve hidden surface removal problems. But they are now used for a wide variety of applications, including set operations in solid modeling, visibility preprocessing for interactive walkthroughs, shadow generation, and cell decomposition methods in robotics, to name only a representative sample [2, 6, 7, 12, 13, 17]. The *size* of a BSP is the total number of pieces the input objects are partitioned into by the BSP, and it is a measure of the *fragmentation* caused by the partition. Because BSPs are often used to decompose large data sets, their size can be crucial to the performance of the applications that rely on them. Theoretical analyses have therefore focused primarily

*Received by the editors August 19, 2004; accepted for publication (in revised form) February 22, 2005; published electronically July 26, 2005. A preliminary version of this work appeared in the *Proceedings of the* 20th *Annual ACM Symposium on Computational Geometry*, ACM, New York, 2004, pp. 230–238.

http://www.siam.org/journals/sicomp/34-6/44570.html

†Mentor Graphics Corp., 8005 SW Boeckman Road, Wilsonville, OR 97070 (john_hershberger@mentor.com), and Department of Computer Science, University of California at Santa Barbara, Santa Barbara, CA 93106.

‡Department of Computer Science, University of California at Santa Barbara, Santa Barbara, CA 93106 (suri@cs.ucsb.edu). This author's research was partially supported by NSF grant CCR-9901958.

§Department of Mathematics, MIT, Cambridge, MA 02139 (toth@math.mit.edu). The research on this paper was performed while this author was a postdoctoral fellow at the University of California at Santa Barbara.

on BSP algorithms that produce small size partitions.

A theoretical study of BSPs began in earnest with two influential papers by Paterson and Yao [14, 15]. In two dimensions, Paterson and Yao gave an $O(n)$ size BSP construction for orthogonal (axis-parallel rectangular) objects, and an $O(n \log n)$ size construction for general polygons with a total of $n$ edges. Recently, Tóth [18] showed an almost-matching lower bound of $\Omega(n \log n / \log \log n)$ for arbitrary polygons in the plane. In three dimensions, Paterson and Yao gave an $O(n^{3/2})$ size BSP construction for orthogonal boxes, and an $O(n^2)$ size construction for arbitrarily oriented polyhedra with a total of $n$ edges. They also gave lower bound constructions matching these upper bounds.

The results of Paterson and Yao have been extended and improved in several important directions, yet the complexity of BSPs in three and higher dimensions is still not fully understood. We briefly review the previous research most relevant to our work. De Berg shows that any *uncluttered* scene of $n$ objects in $d$-space admits a linear size BSP [4]; informally, "uncluttered" means that any cubical region intersecting more than a constant number of objects must include a vertex of the bounding box of one of the objects. For instance, a collection of hypercubes (or, more generally, of *fat* objects) is uncluttered. Such an assumption may be practical in some situations, but it seems quite restrictive in general.

Paterson and Yao [14] consider the complexity of BSPs for 1-dimensional objects (line segments) in $d$-space, and show that a worst-case set of $n$ axis-aligned line segments requires a BSP of size $\Theta(n^{d/(d-1)})$ for $d > 2$. Dumitrescu, Mitchell, and Sharir [8] give an upper bound of $O(n^{d/(d-k)})$ for the BSP size of disjoint $k$-dimensional orthogonal objects in $d$-space for $1 \leq k < d$. This bound is asymptotically tight for $k < d/2$. The only known tight bound for orthogonal $k$-flats in $d$-space with $k \geq d/2$ is the case $d = 4$ and $k = 2$, where a $\Theta(n^{5/3})$ bound has been shown [8].

The general $O(n^{d/(d-k)})$ upper bound does not say anything about full-dimensional boxes. Berman, DasGupta, and Muthukrishnan [5] show that every set of $n$ axis-aligned rectangles in the plane has a BSP of size at most $3n$. The best known lower bound, $\frac{7}{3}n - o(n)$, is due to Dumitrescu, Mitchell, and Sharir [8]. If the rectangles tile the plane, however, Berman, DasGupta, and Muthukrishnan [5] prove an upper bound of $2n - 1$, matching a lower bound of $2n - o(n)$ by Dumitrescu, Mitchell, and Sharir [8] (originally designed for axis-parallel line segments). For full-dimensional axis-aligned boxes in $d$-space, $d \geq 2$, one can obtain an upper bound of $O(n^{d/2})$ by using the result of [8] in conjunction with an observation of Paterson and Yao [15] that a set of $n$ full-dimensional boxes in $d$-space has the same BSP complexity in certain cases as the set of their $(d-2)$-dimensional faces.

Agarwal et al. [1] consider BSPs of 2-dimensional *fat rectangles* in 3-space—each rectangle has sides parallel to the axes and a constant aspect ratio. Agarwal et al. [1] show that a collection of $n$ such rectangles admits a BSP of size $n2^{O(\sqrt{\log n})}$. This bound was recently improved to $O(n \log^8 n)$ by Tóth [19].

**Our results.** In an attempt to understand the true complexity of $d$-dimensional BSPs, we consider a natural but restricted setting: *orthogonal subdivisions.* An orthogonal subdivision is a collection of interior-disjoint rectilinear boxes that fill their containing space. (That is, a subdivision is a space-filling packing of boxes.) Our interest in subdivisions is motivated by the observation that all the known lower bound constructions involve intertwined rod-like objects that create many "holes." This raises the following natural question: What is the complexity of the BSP for polygonal scenes in which the complement space can be "tiled" with few boxes, say, a linear number? Our main result is the following theorem.

Theorem 1.1. *Given an orthogonal subdivision with $n$ boxes in $d$-space, we can construct a BSP for it of size $O(n^{(d+1)/3})$ for any $d \geq 2$.*

Thus, for collections of orthogonal objects that fill their containing space, we are able to improve the worst-case bound on the BSP size from $O(n^{d/2})$ to $O(n^{(d+1)/3})$. As a corollary, for any collection of $n$ orthogonal boxes whose complement space can be tiled with $m$ boxes, there exists a BSP of size $O((n+m)^{(d+1)/3})$. Our subdivision BSP can be constructed in time $O(K \log n)$, where $K$ is the output size. We also exhibit a lower bound construction for an important class of BSPs: An *axis-aligned BSP* is a BSP in which every cutting hyperplane is orthogonal to one of the axes. We describe a subdivision that requires an axis-aligned BSP of size $\Omega(n^{\beta(d)})$ in $\mathbb{R}^d$, where $\beta(d)$ converges to $(1+\sqrt{5})/2$ as $d$ goes to infinity. The value of $\beta(d)$ is $4/3$ for $d = 3$, and thus our upper bound is tight in 3-space.

Our BSP algorithms are quite simple and therefore easy to implement. Their key component is a round robin partitioning scheme (in which cutting hyperplanes orthogonal to the coordinate axes are selected in a round robin order). They follow a conventional scheme—a round robin partitioning phase followed by efficient BSP construction in the terminal regions. Such a two-phase partitioning framework has been used earlier in several papers [1, 8], but in each case it requires problem-specific insights to find the right stopping rule for the round robin phase as well as an analysis for the terminal case. In our case, we show that "round robin cutting until no interior $(d-3)$-dimensional face remains" is a good stopping rule. One tricky part of analyzing this round robin scheme is that each cut may also *increase* the number of $(d-3)$-faces. The second phase requires efficient BSPs for regions that do not contain a $(d-3)$-face. To this end, we prove the following theorem, which may have independent appeal.

Theorem 1.2. *Consider a box $R$ in $d$-space and a subdivision of $R$ into $n$ boxes such that no $(d-3)$-dimensional face of any box intersects the interior of $R$. Then there is a BSP of size $O(n)$ for this subdivision.*

## 2. Geometric preliminaries.

A *$d$-dimensional box $B$* is the cross product of $d$ real-valued intervals. Given a box $R$ and a set of boxes $S = \{B_1, \dots, B_n\}$, we say that $S$ is a *subdivision* of $R$ if each $B_i$ lies in $R$, the union of the $B_i$'s covers $R$, and the $B_i$'s have pairwise disjoint interiors. Thus, an orthogonal subdivision of $R$ is a packing by boxes that completely fills $R$. In this paper, we consider BSPs for $d$-dimensional box subdivisions only.

A BSP for a problem instance $(R, S)$ partitions $R$ with a hyperplane into subboxes $R_1$ and $R_2$ and recursively solves the problems $(R_1, S_1)$ and $(R_2, S_2)$, where $S_i = S \cap R_i$ is the set of fragments of the input objects contained in $R_i$ for $i = 1, 2$. The partitioning stops when every subproblem contains at most one object (fragment). A BSP is naturally modeled as a binary tree: the root corresponds to the problem $(R, S)$, and its two children correspond to the subproblems $(R_1, S_1)$ and $(R_2, S_2)$. Every internal node stores the splitting hyperplane for the corresponding problem; the leaf nodes correspond to the regions in the final partition. See Figure 1 for a simple example in two dimensions. For ease of reference, we will often call $R$ the *container box* for the problem instance. Thus, $R_i$ is the container box for the $i$th subproblem, for $i = 1, 2$, produced at the root.

*Free cuts* are important for the construction of small BSPs. A free cut is a hyperplane that separates the object set without splitting any object. When a subproblem $(R, S)$ has a free cut, it is always worth splitting the subproblem with the free cut, since the split does necessary work—separating objects that must be separated by

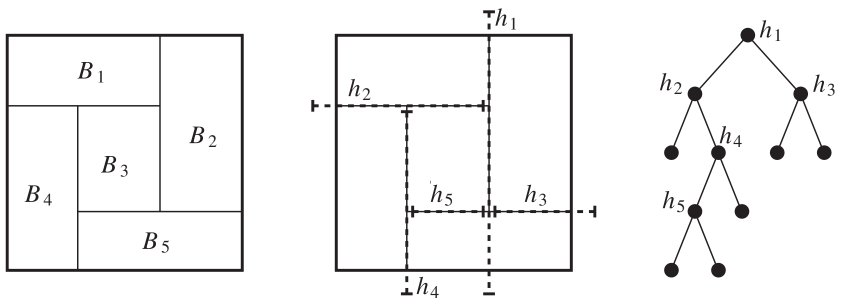


FIG. 1. *An orthogonal subdivision $S$ (left), a BSP for $S$ (middle), and the corresponding binary tree (right).*

the BSP—without increasing the complexity of the subproblems (and hence the final BSP size). In Figure 1, the cuts along $h_2$, $h_3$, $h_4$, and $h_5$ are free cuts.

A $d$-dimensional box $B$ has $2^d$ vertices, $d2^{d-1}$ edges, and $2d$ facets. $B$ also has many faces of intermediate dimensions $j$ for $2 \leq j \leq d-2$. A *$k$-face* of $B$ is a $k$-dimensional box, and it can be characterized as follows: a $k$-face of $B$ is obtained from the cross product $B$ by fixing $(d-k)$ coordinates at one of the two extremes of the corresponding intervals; the remaining $k$ coordinates maintain the same extent as $B$. Thus, vertices are the 0-faces, edges are the 1-faces, and facets are the $(d-1)$-faces of $B$. It is easy to see that every $k$-face of the box $B$, for $0 \leq k < d$, lies on the boundary of $B$. Our algorithm exploits the structure of boxes whose extents in certain dimensions match those of the container box. We therefore introduce the notions of *pass-through* and *$k$-rod*.[1]

Consider a container box $R$ and a box $B$ in a subdivision of $R$. We say that $B$ is *pass-through* for $R$ in dimension $j$ if the extent of $B$ in dimension $j$ equals that of $R$. We say that box $B$ is a *$k$-rod* in $R$ if $B$ is pass-through in exactly $k$ dimensions. If $B$ is a $k$-rod in $R$, then its *orientation* $\sigma(B)$ is the set of the $k$ dimensions in which $B$ is pass-through for $R$. Whenever the container box $R$ is clear from the context, we will simply say that $B$ is a $k$-rod, without mentioning $R$. Figure 2 illustrates $k$-rods in three dimensions, for $k = 1, 2$.

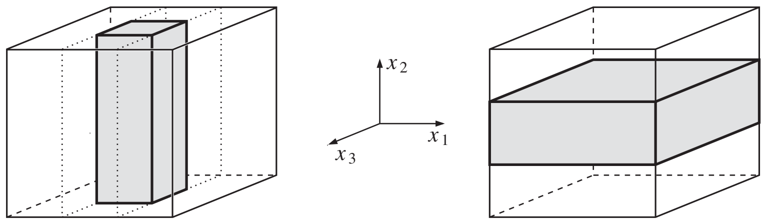


FIG. 2. *A 1-rod of orientation $\{x_2\}$ on the left and a 2-rod of orientation $\{x_1, x_3\}$ on the right.*

Suppose $B$ is a rod in a box subdivision of some container $R$. If $B$ is a $d$-rod, then, of course, $B$ fills up $R$, and it is the only box in $R$. If $B$ is a $(d-1)$-rod, with orientation $\sigma(B) = \{x_1, x_2, \ldots, x_{d-1}\}$, then at least one of the two facets of $B$ orthogonal to the $x_d$-axis intersects the interior of $R$, separating it into two parts. A

[1]The notion of pass-through also appears in [8]. Our analysis, however, requires a more detailed investigation into the properties of rods.

cut along this facet is a free cut for the subdivision of $R$. For instance, in the right half of Figure 2, the box $B$ is a 2-rod in 3-space, and there are free cuts along its top and bottom faces. We summarize this fact for future reference.

LEMMA 2.1. *If a box subdivision contains a $(d-1)$-rod, then there is a free cut along one of the facets of that rod.*

We next establish an elementary but useful characterization of $k$-rods: a box $B$ is a $k$-rod in a container $R$ *if and only if* every $j$-face of $B$ is disjoint from the interior of $R$, for all $j < k$. For instance, in Figure 2 (right), $B$ is a 2-rod, and all of its vertices and edges (0- and 1-faces) lie on the boundary of the container box. We call a face an *interior face* if it intersects the interior of the container box.

LEMMA 2.2. *Suppose $B$ is a box in the subdivision of $R$. Then $B$ is a $k$-rod if and only if $B$ has no interior $j$-faces for $j < k$ and at least one interior $j$-face for every $j$ such that $k \leq j \leq d$.*

*Proof.* Consider a $k$-rod $B$. Let $F$ be one of its $j$-faces, where $j < k$. By definition, $F$ has a fixed coordinate in $(d-j)$ dimensions and the same extent as $B$ in the remaining $j$ dimensions. Because $B$ is a $k$-rod, it is pass-through for $R$ in $k$ dimensions, and $k \geq j+1$. Thus, at least one of the fixed coordinates of $F$ is at one of the extremes of a pass-through dimension of $B$, and so $F$ lies on the boundary of $R$. Now consider $j$ such that $k \leq j \leq d$. Choose $F$ to be a $j$-face whose varying coordinates include all the pass-through directions of the $k$-rod $B$, and whose fixed coordinates are chosen inside the corresponding intervals of the cross product of the container box $R$ (possible because $B$ is pass-through in none of those $d-j \leq d-k$ dimensions). By construction, this $j$-face $F$ intersects the interior of $R$. ∎

## 3. An upper bound in $\mathbb{R}^3$.

Our first result is an optimal BSP for 3-dimensional subdivisions. Our BSP has worst-case size $O(n^{4/3})$, which is optimal because a modified Paterson–Yao construction gives a matching lower bound for orthogonal subdivisions (see section 6). The 3-dimensional bound is central to our main result because our algorithm for constructing the $d$-dimensional BSP uses projection of the $d$-dimensional subdivision onto an appropriate 3-space (after a suitable number of round robin cuts).

Our algorithm (Algorithm 3-BSP) is a round robin partitioning scheme that iteratively slices the subdivision by planes passing through interior box vertices, producing a collection of smaller subdivisions, each of which ultimately contains only $j$-rods, for $j \geq 1$. A similar round robin scheme is also used by Murali [11] in his construction of BSPs for fat axis-aligned rectangles in $\mathbb{R}^3$. Interestingly, while Murali also achieves an $O(n^{4/3})$ size BSP, that complexity is suboptimal for his problem. In our case, the round robin algorithm produces an optimal BSP.

The partition tree $T$ created by Algorithm 3-BSP is *not* a BSP—the cells output by the algorithm may contain multiple boxes. These terminal regions, however, do not contain any vertices of the subdivision. We can refine each such box into a proper BSP, with only a constant factor increase in complexity, and append the tree associated with this refinement to each leaf of $T$. Therefore, the key to efficiency is bounding the total complexity of all the regions produced by the slicing procedure.

Suppose we have a container box $R$ and its subdivision $S = \{B_1, B_2, \ldots, B_n\}$ into $n$ boxes, so that there are $m$ vertices in the interior of $R$; clearly, $m \leq 8n$. Our partitioning scheme cuts the subdivision using planes normal to each of the three axes in turn, cycling through the three possible orientations in a round robin fashion. In the following pseudocode description of our algorithm, we use the term *median $x_j$ plane* of a point set to denote the plane normal to the $x_j$-axis and passing through the median $x_j$ coordinate of the point set.

ALGORITHM 3-BSP.

```
• Input is the box subdivision $S = \{B_1, B_2, \dots, B_n\}$ of a container box $R$.
• Initialize $i = 0$, and $\mathcal{C}_0 = \{R\}$.
• While there is a container box $C \in \mathcal{C}_i$ with a subdivision vertex in its interior, do
    1. For each container box $C \in \mathcal{C}_i$ with at least one vertex of the subdivision
       in its interior, split $C$ by the median $x_p$ plane of the vertices in the
       interior of $C$, where $p = 1 + (i \bmod 3)$.
    2. Apply all possible free cuts.
    3. Let $\mathcal{C}_{i+1}$ be the set of all container boxes resulting from these cuts, and
       set $i := i + 1$.
• Return $\mathcal{C}_i$.
```

Using an argument similar to one of Paterson and Yao [15] and Murali [11], we derive an upper bound on the total number of box fragments generated by our algorithm by considering how many times the edges of the original subdivision $S$ are cut.

LEMMA 3.1. *Consider the partition tree $T$ generated by Algorithm 3-BSP. At depth $i$ in $T$, there are $O(n2^{i/3})$ fragments of the original edges of $S$.*

*Proof.* Each edge is parallel to one of the three directions $x_1$, $x_2$, and $x_3$. It can be cut only by planes orthogonal to its direction. Since our splitting planes cycle through the three directions, an edge can be cut at every third level of $T$. Thus, the number of fragments of a given original edge at most doubles at every third level. Since the number of edges in the subdivision at the root of $T$ is $O(n)$, the total number of edge fragments at depth $i$ is $O(n2^{i/3})$. $\square$

LEMMA 3.2. *Suppose $S$ is a 3-dimensional box subdivision with $n$ boxes, and $m = O(n)$ vertices lie in the interior of the container box. Then the partition created by Algorithm 3-BSP produces $O(nm^{1/3}) = O(n^{4/3})$ fragments of the input boxes.*

*Proof.* We analyze the algorithm in *rounds* of three consecutive steps: round $j$ corresponds to the steps $i = 3j, 3j+1, 3j+2$. In one round, cuts are made in all three directions, and a box of a subproblem can be split into at most eight pieces.

We observe that if a box is split into a subproblem, then at least one edge of the box must be interior to the container box for this subproblem. This is because only a 3-rod or a 2-rod can have all its edges on the boundary of the container box (Lemma 2.2), but no 3-rods are ever split and all 2-rods are eliminated in step 2 by free cuts. Lemma 3.1 implies that $O(n2^j)$ box fragments can be further partitioned at round $j$. Since there is no vertex in the interior of any container cell after $\lceil \log m \rceil$ steps, the algorithm terminates in $\lceil \frac{1}{3} \log m \rceil$ rounds. The number of box fragments produced is

$$O\left(n \cdot \sum_{j \le \lceil \frac{1}{3} \log m \rceil} 2^j\right) = O\left(n \cdot 2^{(\log m)/3}\right) = O\left(nm^{1/3}\right) = O\left(n^{4/3}\right). \qquad \square$$

All that remains now is to show that the terminal cells output by Algorithm 3-BSP can be refined into linear size BSPs. The interior of each of these terminal cells is empty of subdivision vertices and, by Lemma 2.2, every box restricted to these containers is a $k$-rod, $k \ge 1$. We first establish a useful technical lemma for the case where all subdivision boxes are 1-rods.

Lemma 3.3. *If all boxes in a* 3*-dimensional subdivision are* 1*-rods, then taken together the* 1*-rods have at most two distinct orientations.*

*Proof.* Suppose to the contrary that $A$, $B$, and $C$ are three 1-rods with three distinct orientations. Let $\ell_A$, $\ell_B$, and $\ell_C$ be lines running down the centers of $A$, $B$, and $C$, respectively, parallel to their rods' orientations. These are three axis-parallel skew lines. Let $P_{AB}$ be the axis-aligned plane containing $\ell_A$ and intersecting $\ell_B$ (see Figure 3). Define $P_{BC}$ and $P_{CA}$ similarly. Note that $P_{AB}$ does not intersect $\ell_C$ because it is parallel to it. The point $P_{AB} \cap \ell_B$ lies in the interior of $B$—$P_{AB}$ lies between the planes supporting the two opposite faces of the container box that $\ell_B$ pierces. Now $P_{AB} \cap P_{BC}$ is a line parallel to $\ell_A$ that intersects $\ell_B$. It follows that $P_{AB} \cap P_{BC}$ is disjoint from $A$ and intersects $B$. Similar claims hold for $P_{BC} \cap P_{CA}$ and $P_{CA} \cap P_{AB}$.

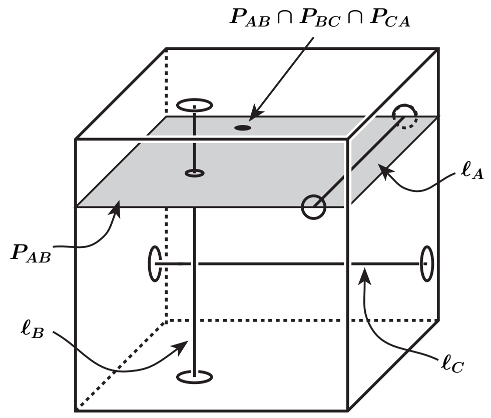


Fig. 3. 1*-rods with three different orientations imply the existence of an interior vertex.*

Let $p$ be the point $P_{AB} \cap P_{BC} \cap P_{CA}$. Point $p$ is disjoint from $A \cup B \cup C$, and $p$ also lies in the interior of the container box, since it is connected by three axis-parallel lines to the points $P_{AB} \cap \ell_B$, $P_{BC} \cap \ell_C$, and $P_{CA} \cap \ell_A$, all of which lie in the interior of the container box. Now we claim that the box containing $p$ cannot be a 1-rod in this subdivision: for each of the axis-parallel directions there is a line (e.g., $P_{AB} \cap P_{BC}$) that intersects one of $A$, $B$, and $C$, and hence the box containing $p$ cannot touch any pair of opposite faces of the container box. This completes the proof. $\square$

Lemma 3.4. *Consider a set of boxes $S = \{B_1, B_2, \ldots, B_n\}$ that forms a subdivision of a* 3*-dimensional container box $R$. If none of the vertices of any box $B_i$ is in the interior of $R$, then there is a BSP of size at most $2n - 1$ for $S$.*

*Proof.* We proceed by induction on $n$. The base case, $n = 1$, is trivial. Now suppose that Lemma 3.4 holds for every $n'$, $1 \leq n' < n$. If no 0-face of any box $B$ lies in the interior of $R$, then every box is a rod by Lemma 2.2. A 3-rod completely fills the container, and thus $n = 1$ and no BSP cuts are needed. If there is a 2-rod, then there is a free cut by Lemma 2.1: We split $R$ into two subdivisions along the free cut such that each has fewer than $n$ boxes, and induction completes the proof. We may assume, therefore, that all boxes are 1-rods. By Lemma 3.3, all rods in the subdivision have at most two different orientations. We handle the two cases below separately.

If the rods in the subdivision have two different orientations, we show that there is a free cut, and the proof then follows by induction, since each subproblem produced by the free cut is strictly smaller than $n$. Suppose, without loss of generality, that the orientation of every rod is either $\{x_1\}$ or $\{x_2\}$. Observe that all the 1-rods properly intersecting a hyperplane $H$ orthogonal to $x_3$ must have the same orientation, or else two rods with different orientations would intersect on $H$. (This observation is equivalent to applying Lemma 4.1 to the 2-dimensional subdivision on $H$.) Now consider a point $p$ on the common boundary of two 1-rods of different orientations, and let $H$ be a hyperplane passing through $p$ and orthogonal to $x_3$. Any (closed) box of $S$ intersecting $H$ lies on one side of $H$ because there are 1-rods of different orientations on the two sides of $H$. Every box intersecting $H$ has a face along $H$, and therefore $H$ is a free cut (see Figure 4).

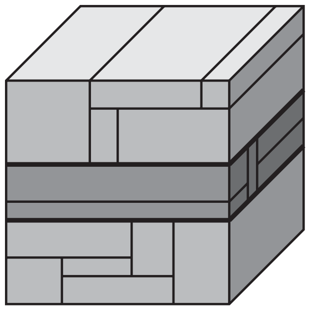


FIG. 4. 1-*rods with two different orientations imply the existence of a free cut.*

If all the 1-rods in the container have the same orientation, then we project them onto the plane orthogonal to their orientation. This gives a 2-dimensional subdivision, which has a 2-dimensional BSP of size at most $2n-1$ by a result of Berman, DasGupta, and Muthukrishnan [5]. Extending each linear cut of the 2-dimensional BSP into a planar cut parallel to the rod orientation, we obtain a BSP for $S_A$ of size at most $2n-1$. $\square$

Thus, combining Lemmas 3.2 and 3.4, we get the main result of this section.

THEOREM 3.5. *A 3-dimensional box subdivision with $n$ boxes and $m$ interior vertices has a BSP of size $O(nm^{1/3}) = O(n^{4/3})$. We can construct such a BSP in time $O(\min(nm^{1/3}, K \log n))$, where $K$ is the size of the final BSP.*

The implementation that achieves the $O(\min(nm^{1/3}, K \log n))$ construction time is quite simple. We maintain all the extents of all the boxes in all containers. It is easy to determine from the extents which boxes are split into two, where the median planes lie, and which faces are free cuts. If we sort the extents initially in all dimensions, we can find medians trivially in linear time and can create sorted extent lists for the child containers in the same time. We spend $O(n \log n)$ time for the initial sorting and $O(K)$ time for each of the $O(\log n)$ steps. Because $K \geq n$, this gives a running time of $O(K \log n)$.

The same implementation also has a running time of $O(nm^{1/3})$—without a logarithmic factor. By Lemma 3.1, the number of box fragments at step $i$ is $O(n2^{i/3})$. The time for step $i$ is therefore $O(n2^{i/3})$ as well. Because the time per step increases geometrically, the last step dominates the running time, giving a bound of $O(nm^{1/3})$.

The size upper bound of $O(n^{4/3})$ in Theorem 3.5 is tight in the worst case: In section 6, we exhibit a 3-dimensional subdivision requiring a BSP of size $\Omega(n^{4/3})$.

**4. The structure of rods in $\mathbb{R}^d$.** In order to extend our 3-dimensional BSP algorithm to higher dimensions, we need two key ingredients: (1) a suitable stopping rule for our round robin algorithm, that is, one that does not result in too much fragmentation, and (2) a linear size BSP construction for the cells obtained when the round robin phase ends. In three dimensions, the correct stopping rule was the elimination of interior vertices. In $d$ dimensions, we show that the *elimination of interior* $(d-3)$*-dimensional faces* is a good stopping rule. The analysis of the total fragmentation caused by this stopping rule as well as the construction of BSPs in terminal cells requires a better understanding of higher-dimensional geometry. In this section, we establish two key facts about the $d$-dimensional subdivisions that are central to our analysis.

LEMMA 4.1. *Consider two boxes $B_1, B_2$ that are rods for their container box $R$ in d-space. Let $D_1, D_2 \subset \{1,2,\dots,d\}$ be the sets of dimensions in which $B_1$ and $B_2$, respectively, are pass-through. If $D_1 \cup D_2 = \{1,2,\dots,d\}$, then $B_1$ and $B_2$ have intersecting interiors.*

*Proof.* Let $\ell_i$ be the extent of dimension $x_i$ *common* to $B_1$ and $B_2$. Since in each dimension at least one of the rods is pass-through, $\ell_i$ is nonempty for every $i$. Thus a full-dimensional box defined by the cross product $\ell_1 \times \cdots \times \ell_d$ lies in the common interior of $B_1$ and $B_2$. □

This lemma implies that, for any two boxes $B_i, B_j$ in a subdivision, there is at least one direction in which neither is a rod. With this basic property, we next classify the set system of all possible orientations of $(d-2)$-rods in a box subdivision. The $(d-2)$-rods are especially important to us because our ultimate goal is to consider subdivisions without any interior $(d-3)$-faces. Recall that a set system is a pair $(X, \mathcal{F})$, where $X$ is a ground set, and $\mathcal{F}$ is a family of subsets of $X$. We consider the set system $(D, \mathcal{F})$, where $D = \{1,2,\dots,d\}$, and $|F| = d-2$ for all $F \in \mathcal{F}$. We are interested in the following two configurations:

- **Cycle configuration:** $\mathcal{F}$ is a *cycle configuration* if $\mathcal{F}$ has at least three sets and there is some $x \in D$ that is not an element of any set of $\mathcal{F}$.
- **Star configuration:** $\mathcal{F}$ is a *star configuration* if $\mathcal{F}$ has at most three sets and they all share a common $(d-3)$ element subset of $D$.

LEMMA 4.2. *Consider a set system $(D, \mathcal{F})$ with $D = \{1,2,\dots,d\}$ and $|F| = d-2$ for every $F \in \mathcal{F}$. If every two sets $F_1, F_2 \in \mathcal{F}$ have $F_1 \cup F_2 \neq D$, then $\mathcal{F}$ is either a star or a cycle configuration.*

*Proof.* Consider the set system $G = (D, E)$, $E = \{D \setminus F : F \in \mathcal{F}\}$. Every $e \in E$ has two elements, and so $G$ is a graph with vertex set $D$ and edge set $E$. The condition that $F_1 \cup F_2 \neq D$ for any $F_1, F_2 \in \mathcal{F}$ is equivalent to saying that any two distinct edges of $E$ are adjacent. $\mathcal{F}$ is a cycle configuration if and only if $G$ is a star graph (all edges are incident to a common vertex) with at least three edges. $\mathcal{F}$ is a star configuration if and only if $G$ is a subgraph of a triangle.

To prove that $\mathcal{F}$ is either a cycle or a star configuration, it suffices to show that every graph with pairwise adjacent edges is either a triangle or a star graph. Elementary graph theory completes the proof: If $|E| < 3$, then $G$ is trivially a star graph, so let us assume that $|E| \geq 3$. If there is a triangle in $G$, then $|E| = 3$, and $E$ forms a triangle because any fourth edge would be nonadjacent to at least one edge of the triangle. Now assume that $G$ is triangle-free and consider two adjacent edges, $e_1 = \{a, b\}$ and $e_2 = \{a, c\}$, $b \neq c$. Since any $e_3 \in E \setminus \{e_1, e_2\}$ is adjacent to both $e_1$ and $e_2$, but $e_3 \neq \{b, c\}$, $e_3$ must be incident to $a$. So every edge is incident to $a$, and $G$ is a star graph. □

This lemma turns out to be a critical technical piece in constructing linear size BSPs in higher dimensions whenever the subdivision is free of $(d-3)$-dimensional faces. A 4-dimensional version of our star-cycle property was observed by Dumitrescu, Mitchell, and Sharir [8].

## 5. An upper bound in $\mathbb{R}^d$.

We begin by showing that if a $d$-dimensional subdivision is free of all interior $(d-3)$-faces, then it admits a linear size BSP. This fact ensures that if we use the stopping rule "recurse until no interior $(d-3)$-dimensional faces remain," the second phase of the algorithm incurs only a linear additional fragmentation.

LEMMA 5.1. *Consider a set of boxes $S = \{B_1, B_2, \ldots, B_n\}$ that forms a subdivision of a $d$-dimensional container box $R$. If none of the $(d-3)$-faces of any box $B_i$ is in the interior of $R$, then there is a BSP of size at most $2n-1$ for $S$.*

*Proof.* Our proof is by induction on $n$. The base case is $n = 1$, for which there is nothing to prove. If any of the boxes of $S$ is a $(d-1)$-rod, then we can make a free cut along a facet of this box by Lemma 2.1; each of the subproblems on either side of the cut contains strictly fewer than $n$ boxes, so our lemma follows by induction. Lemma 2.2 tells us that all boxes of $S$ are $(d-2)$-rods. By Lemmas 4.1 and 4.2, the orientations of these $(d-2)$-rods form either a star or a cycle configuration. We handle these two cases separately.

First, suppose that the orientations of the rods form a cycle configuration. Without loss of generality, let us assume that none of the $(d-2)$-rods is pass-through in direction $x_d$. We will show that there is a free cut orthogonal to the $x_d$-axis, and the rest of the proof then follows by induction. We observe that all the $(d-2)$-rods properly intersecting a hyperplane $H$ orthogonal to $x_d$ must have the same orientation; otherwise Lemma 4.1 is violated for the $(d-1)$-dimensional subdivision on $H$. Now consider a point $p$ on the common boundary of two $(d-2)$-rods of different orientations, and let $H$ be a hyperplane passing through $p$ and orthogonal to $x_d$. Any (closed) box of $S$ intersecting $H$ lies on one side of $H$ because there are $(d-2)$-rods of different orientations on the two sides of $H$. Every box intersecting $H$ has a facet along $H$, and therefore $H$ is a free cut.

Next, suppose that the orientations of the rods form a star configuration. Without loss of generality, assume that all boxes are pass-through in $(d-3)$ dimensions $x_4, x_5, \ldots, x_d$. We project the boxes of $S$ onto the subspace spanned by the remaining three axes, using the map $\pi : (x_1, x_2, \ldots, x_d) \to (x_1, x_2, x_3)$. Any BSP for the projection corresponds to a BSP of equal size for $S$, since the preimage of a plane $h$ in the projection is a hyperplane $H$ in $\mathbb{R}^d$, and $H$ cuts only those $(d-2)$-rods that are the preimages of the 3-dimensional boxes cut by $h$. Observe that the subdivision $\pi(S)$ does not have any interior vertices (interior in $\mathbb{R}^3$), because an interior vertex $v$ corresponds to an interior $(d-3)$-face in $S$, namely, $\pi^{-1}(v)$. By Lemma 3.4, there is a BSP of size at most $2n-1$ for the projected subdivision which, when lifted to $\mathbb{R}^d$, gives a BSP of the same size for $S$. □

We are now ready to describe our BSP algorithm for $d$-dimensional subdivisions. In three dimensions, our round robin cuts were made along planes that evenly partition the set of interior vertices. In $d$ dimensions, we choose hyperplanes that "evenly split" the interior $(d-3)$-dimensional faces. Unfortunately, these median hyperplanes can also introduce new $(d-3)$-dimensional faces by splitting some old ones. For instance, in 4-space, a cutting hyperplane can split many 1-faces (line segments). Our fragmentation lemma (Lemma 5.2) below will bound the total number of box fragments created in the round robin phase. We need to introduce a bit of notation first to discuss the objects used in the algorithm.

Given a $d$-dimensional box $B$, let $f_k(B)$ denote the set of $k$-faces of $B$, where $0 \le k \le d$. We use the notation $f_k(S)$ to refer to the (multi) set of $k$-faces in all the boxes of $S = \{B_1, B_2, \ldots, B_n\}$; that is, $f_k(S)$ contains multiple copies of a face if that face is incident to multiple boxes. Finally, given a container box $R$, we use the notation $f_k(R, S)$ to denote the (fragments of) interior $k$-faces of $f_k(S)$ (faces that intersect the interior of $R$). In our $d$-dimensional BSP scheme, the elements of the set $f_{d-3}(R, S)$ will play the role that vertices played in Algorithm 3-BSP.

Algorithm $d$-BSP.

```
• Input is a $d$-dimensional box subdivision $S = \{B_1, B_2, \ldots, B_n\}$ of a container
  box $R$.
• Initialize $j = 0$, and $\mathcal{C}_0 = \{R\}$.
• While there is a container box $C \in \mathcal{C}_j$ with nonempty $f_{d-3}(C, S)$, do
    1. For each container box $C \in \mathcal{C}_j$ with nonempty $f_{d-3}(C, S)$, do
        (a) Set $\mathcal{D}_1 = \{C\}$ and let $V$ be the multiset of vertices of all faces in
            $f_{d-3}(C, S)$.
        (b) For $i = 1$ to $d$, do
              i. Split every $D \in \mathcal{D}_i$ by the median $x_i$ hyperplane of the multiset
                 $V \cap D$.
             ii. Apply all possible free cuts.
            iii. Let $\mathcal{D}_{i+1}$ be the set of all container boxes resulting from these
                 cuts.
             iv. Remove from $V$ all vertices on the splitting planes used in the
                 first two steps.
    2. Let $\mathcal{C}_{j+1}$ be the set of all container boxes resulting from these cuts, and
       set $j := j + 1$.
• Return $\mathcal{C}_j$.
```

Algorithm $d$-BSP constructs a partition tree $T$ whose leaves correspond to regions that do not contain any fragment of a $(d-3)$-face of the subdivision $S$. By Lemma 5.1, each of these regions can be refined into a BSP of size linear in the number of box fragments contained in it. Thus, it suffices to bound the total complexity of all the box fragments produced by Algorithm $d$-BSP.

Lemma 5.2. *Given a $d$-dimensional box subdivision with $n$ boxes, Algorithm $d$-BSP produces $O(n^{(d+1)/3})$ box fragments.*

*Proof.* Let us define $m = |f_{d-3}(R, S)|$. We call a *round* of the algorithm the work done between increments of $j$. First we show that Algorithm $d$-BSP terminates in $\lceil (\log m)/3 \rceil$ rounds, and then we bound the number of box fragments produced in $\lceil (\log m)/3 \rceil$ rounds.

During a round, every $(d-3)$-face $F$ of a subproblem $(C, S)$ can be cut into at most $2^{d-3}$ fragments—it is cut at most once for each of the $(d-3)$ directions orthogonal to $F$. We focus on the fragments of $F$ that are not contained in any splitting hyperplane of the round. We argue that at least one vertex of every fragment of $F$ is a vertex of $F$ that does not lie on any of the splitting hyperplanes of the round. A fragment is itself a $(d-3)$-face, and so it is the cross product of three fixed coordinates and $(d-3)$ nontrivial intervals. Each nontrivial interval contains at least one of the two endpoints of the corresponding original interval of $F$ (because there is at most one cut orthogonal to the interval's direction). The cross product of the three fixed coordinates and the $(d-3)$ original interval endpoints is a vertex of $F$ but does not lie on any splitting hyperplane of the round.

The container box $C$ is divided into subboxes during the round, each containing at most $2^{d-3} \cdot |f_{d-3}(C,S)|/2^d = |f_{d-3}(C,S)|/8$ vertices of $V$. As noted, each face fragment produced by the round is incident to a vertex from $V$, and so $|f_{d-3}(C',S)| \le |f_{d-3}(C,S)|/8$ for each subcontainer $C'$ produced. By induction, the number of interior $(d-3)$-faces at the beginning of round $j$ is at most $m/8^j$. Therefore, after $\lceil \log_8 m \rceil = \lceil (\log m)/3 \rceil$ rounds, no $(d-3)$-face intersects the interior of any subproblem's container box, and the algorithm terminates.

Now consider a $(d-2)$-dimensional face of a box $B$. During a round of the algorithm, it can be cut recursively in $(d-2)$ steps. Thus, the total number of fragments of $(d-2)$-faces of $f_{d-2}(R,S)$ at round $j$ is $O(n2^{(d-2)j})$.

Finally, in each round $j$ when a box fragment $B$ in some subproblem is split, the container box for the subproblem must have a fragment of a $(d-2)$-face of $B$ in its interior. This is because only a $d$-rod or a $(d-1)$-rod can have all its $(d-2)$-faces on the boundary of the container by Lemma 2.2. But a $d$-rod is never split and step 1(b)ii eliminates all $(d-1)$-rods. This implies that $O(n2^{(d-2)j})$ box fragments can be further split in round $j$. The algorithm terminates in $\lceil (\log m)/3 \rceil$ rounds, and so the number of box fragments produced by the algorithm is

$$O\left(n \cdot \sum_{j \le \lceil \log m/3 \rceil} 2^{(d-2)j}\right) = O\left(n \cdot 2^{(d-2)(\log m)/3}\right)$$
$$= O\left(nm^{(d-2)/3}\right) = O\left(n^{(d+1)/3}\right). \qquad \square$$

Since $m$ is at most $2^3\binom{d}{3}\cdot n$, our bound in terms of both $n$ and $d$ is $O(d^{d-2}\cdot n^{(d+1)/3})$.

Finally, this result, combined with Lemma 5.1 and standard implementation techniques similar to those used for Algorithm 3-BSP, yields our main theorem.

THEOREM 5.3. *A box subdivision of $n$ boxes in $d$-space, $d \ge 2$, has a BSP of size $O(n^{(d+1)/3})$. The BSP can be constructed in time $O(\min(n^{(d+1)/3}, K \log n))$, where $K$ is the size of the BSP.*

A simple but practical corollary of this theorem is the following: Any collection of $n$ disjoint boxes in $d$-space whose complement space can be tiled with $m$ additional boxes admits a BSP of size $O((n+m)^{(d+1)/3})$.

If we perform Algorithm $d$-BSP with $(d-2)$-faces instead of $(d-3)$-faces (i.e., replacing $f_{d-3}(C,S)$ by $f_{d-2}(C,S)$ everywhere in the algorithm description), then we obtain a BSP for general nonoverlapping boxes in $d$-space: in the resulting partition, no $(d-2)$-face of any input box intersects the interior of any container box, so every such container can be refined to a proper BSP by free cuts. Repeating Lemma 5.2 using $(d-2)$-faces instead of $(d-3)$-faces implies that the modified algorithm terminates in $\lceil (\log m)/2 \rceil$ rounds, where $m = |f_{d-2}(R,S)| \le 2^2\binom{d}{2} \cdot n$. By analogous computation, the total number of box fragments is $O\left(n \cdot 2^{(d-2)(\log m)/2}\right) = O\left(nm^{(d-2)/2}\right) = O\left(n^{d/2}\right)$. This is the best currently known upper bound on the size of a BSP for $n$ nonoverlapping general boxes in $\mathbb{R}^d$. We can summarize this argument in the following theorem.

THEOREM 5.4. *Every set of $n$ nonoverlapping boxes in $d$-space, $d \ge 2$, has a BSP of size $O(n^{d/2})$.*

One interpretation of our results suggests that for a set $S$ of general axis-aligned boxes in $\mathbb{R}^d$, a round robin BSP for their $(d-2)$-dimensional faces is also a BSP for $S$; while for a $d$-dimensional box subdivision $S$, a round robin BSP for only the $(d-3)$-dimensional faces gives a BSP for the input $S$.

**6. Lower bounds.** Paterson and Yao gave a configuration of $3n$ (non–space-filling) axis-aligned rods such that any BSP for the configuration cuts the rods into $\Omega(n^{3/2})$ subcells [15]. (According to Eppstein [9], "Paterson and Yao credit this application of the shape to a personal communication by W. P. Thurston, but the shape itself has been seen before, e.g., in Alan Holden's book *Shapes, Space, and Symmetry* (Columbia Univ. Press 1971), p. 161.") We describe a variant of the Paterson–Yao construction in some detail, because it is the basis of our lower bound for box subdivisions.

The rods of the Paterson–Yao construction belong to three families, each parallel to one of the coordinate axes, and form an interlocking grid. See Figure 5. Without loss of generality, assume that $n = \ell^2$ for some integer $\ell$. The $x$-, $y$-, and $z$-parallel families consist of boxes indexed by $1 \le i, j, k \le \ell$:

$$R_1 = \{r_1(j,k) = [0, 2\ell] \times [2j, 2j+1] \times [2k, 2k+1] : j, k = 0, 1, \dots, \ell - 1\},$$
$$R_2 = \{r_2(i,k) = [2i, 2i+1] \times [0, 2\ell] \times [2k+1, 2k+2] : i, k = 0, 1, \dots, \ell - 1\},$$
$$R_3 = \{r_3(i,j) = [2i+1, 2i+2] \times [2j+1, 2j+2] \times [0, 2\ell] : i, j = 0, 1, \dots, \ell - 1\}.$$

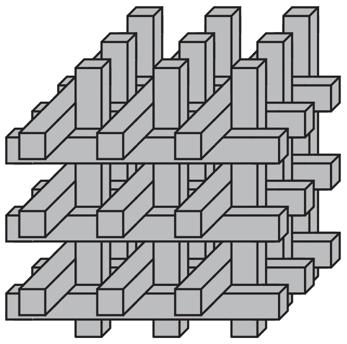


FIG. 5. *The Paterson–Yao lower bound construction, with the rods slightly separated and lengthened to make them easier to see.*

It is straightforward to observe that the rods in each family are disjoint. Furthermore, rods in different families are disjoint, because for each pair of families there is one of the three dimensions ($x$, $y$, or $z$) in which the two families lie in disjoint ranges: for one family, any rod's coordinates lie in the range $[2A, 2A+1]$, for some integer $A$; for the other family, any rod's coordinates lie in $[2B+1, 2B+2]$, for integral $B$. Note also that each grid point $(2i+1, 2j+1, 2k+1)$, for $1 \le i, j, k \le \ell$, is incident to one rod from each of the three families.

For every $i, j, k = 0, 1, \dots, \ell - 1$, consider the box

$$[2i, 2i+2] \times [2j, 2j+2] \times [2k, 2k+2].$$

We call these boxes *junctions*. The junctions are interior-disjoint boxes, and each must be cut by a BSP plane passing through its center, since the BSP must separate the three rods in the neighborhood of the center $(2i+1, 2j+1, 2k+1)$. The first plane that passes through the junction must cut at least one of the three rods incident to the center. Since all the junctions are disjoint, the total number of rod cuts is at least $\ell^3 = n^{3/2}$.

The Paterson–Yao construction is *opaque*, in the sense that every axis-parallel line passing through the container cube

$$[0, 2\ell] \times [0, 2\ell] \times [0, 2\ell]$$

intersects the closure of at least one rod. However, the configuration of rods is not a subdivision: the rods do not fill space. This is easy to prove by observing that the total volume of the container cube is $8\ell^3$, and the total volume of each rod inside is $2\ell$. There are $3\ell^2$ rods, for a total rod volume inside the container cube of $6\ell^3$, which is less than $8\ell^3$. The empty space is distributed in $2\ell^3$ unit cubes. In particular, for each junction for $i, j, k$, the unit cubes

$$[2i, 2i + 1] \times [2j + 1, 2j + 2] \times [2k, 2k + 1]$$

and

$$[2i + 1, 2i + 2] \times [2j, 2j + 1] \times [2k + 1, 2k + 2]$$

are empty. See Figure 6.

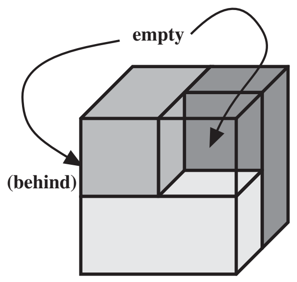


FIG. 6. *A unit cube centered on a grid point in the Paterson–Yao construction.*

**6.1. Lower bound construction for 3-dimensional subdivisions.** Converting the Paterson–Yao configuration into a subdivision requires adding $\Theta(\ell^3)$ new cells—two unit cubes for each junction. Thus the $\Omega(\ell^3)$ lower bound on the BSP complexity becomes trivial—it is linear in the input size. However, with a little more work, we can extend the construction to give a nontrivial lower bound.

THEOREM 6.1. *There is a 3-dimensional subdivision of space into $n$ axis-aligned boxes such that any axis-aligned BSP for the subdivision cuts the boxes into $\Omega(n^{4/3})$ subcells.*

*Proof.* We begin with a Paterson–Yao construction consisting of $3\ell^2$ rods, and then extend it to a subdivision of the container cube $[0, 2\ell]^3$ by adding the $2\ell^3$ unit cubes needed to fill in the empty space between the rods. We then subdivide each rod longitudinally into $\ell$ parallel subrods.

As in the original Paterson–Yao argument, each of the $\ell^3$ junctions must be cut by at least one BSP plane, since the BSP must separate the cells incident to $(2i + 1, 2j + 1, 2k + 1)$. The first plane that cuts a junction must completely cross one of the original rods, and hence it cuts at least $\ell$ subrods. The total number of rod cuts is at least $\ell^3 \times \ell = \ell^4$. The total number of cells in our subdivision is $n = 3\ell^2 \times \ell + 2\ell^3 = 5\ell^3$, and hence the number of subcells produced by the BSP is $\Omega(n^{4/3})$. ☐

**6.2. Constructions for multidimensional subdivisions.** Our multidimensional construction is similar in spirit to the previous construction. This construction is recursive—we use the lower bound construction in $(d-1)$-space to build the subdivision in $d$ dimensions. We begin with an informal description, then formally describe the construction in four dimensions. The construction in $d$ dimensions is a straightforward extension of this construction.

We partition the container box in a grid-like fashion into disjoint regions that we call *junctions* and give a lower bound for the BSP complexity for the fragments of objects within each region. Then we bound the complexity of a BSP for all objects by the sum of complexities over all the regions.

We have $(d-1)$ families of congruent $(d-2)$-rods, whose orientations form a cycle configuration, and a family of 1-rods whose orientation is the complement of the orientations of the $(d-2)$-rods. The rods ensure our complexity bound in each region. (Intuitively, it is efficient to use rods of larger orientations, because a $k$-rod of the container box is a $k$-rod in each junction it intersects.) On the other hand, rods of such orientations cannot fill the container box, and therefore we need to add a number of filler boxes to obtain a box subdivision.

In order to balance the number of rods and the number of filler boxes, we allocate the rods in *bundles* so that the union of a bundle of rods is again a rod with the same orientation. The $(d-2)$-rods in a bundle are all congruent; the $\mathbb{R}^{d-1}$-projection of a bundle of 1-rods, however, is the worst-case box subdivision we obtain in $\mathbb{R}^{d-1}$.

THEOREM 6.2. *There is a* 4*-dimensional box subdivision with n boxes such that the size of any axis-aligned BSP is* $\Omega(n^{13/9})$.

*Proof.* First, we describe an auxiliary construction of $\Theta(\ell^{9/2})$ boxes such that all vertices of the boxes lie on a $2\ell \times 2\ell \times 2\ell \times 3\ell^{3/2}$ grid. The rods of the auxiliary subdivision correspond to the bundles of rods in the final subdivision.

We have $\ell^3$ long 1-rods that are pass-through in $x_4$:

$$R_1 = \{r_1(i,j,k) = [2i, 2i+1] \times [2j, 2j+1] \times [2k, 2k+1] \times [0, 3\ell^{3/2}] : \\ i,j,k = 0, \ldots, \ell-1\}.$$

We also have three families of boxes, each containing $\ell^{5/2}$ 2-rods with orientations $\{x_1, x_2\}$, $\{x_1, x_3\}$, and $\{x_2, x_3\}$, respectively:

$$R_{1,2} = \{r_{1,2}(k,t) = [0, 2\ell] \times [0, 2\ell] \times [2k+1, 2k+2] \times [3t, 3t+1] : \\ k = 0, 1, \ldots, \ell-1;\ t = 0, \ldots, \ell^{3/2}-1\},$$

$$R_{1,3} = \{r_{1,3}(j,t) = [0, 2\ell] \times [2j+1, 2j+2] \times [0, 2\ell] \times [3t+1, 3t+2] : \\ j = 0, \ldots, \ell-1;\ t = 0, \ldots, \ell^{3/2}-1\},$$

$$R_{2,3} = \{r_{2,3}(i,t) = [2i+1, 2i+2] \times [0, 2\ell] \times [0, 2\ell] \times [3t+2, 3t+3] : \\ i = 0, \ldots, \ell-1;\ t = 0, \ldots, \ell^{3/2}-1\}.$$

Figure 7 illustrates the intersection of our auxiliary construction with three (3-dimensional) hyperplanes orthogonal to $x_4$. The intersection of the 1-rods in $R_1$ with every hyperplane consists of $\ell^3$ unit cubes in a grid. The 2-rods of three distinct orientations appear in three different slices. In each slice, congruent 2-rods fill $\ell$ slots between the grid of unit cubes.

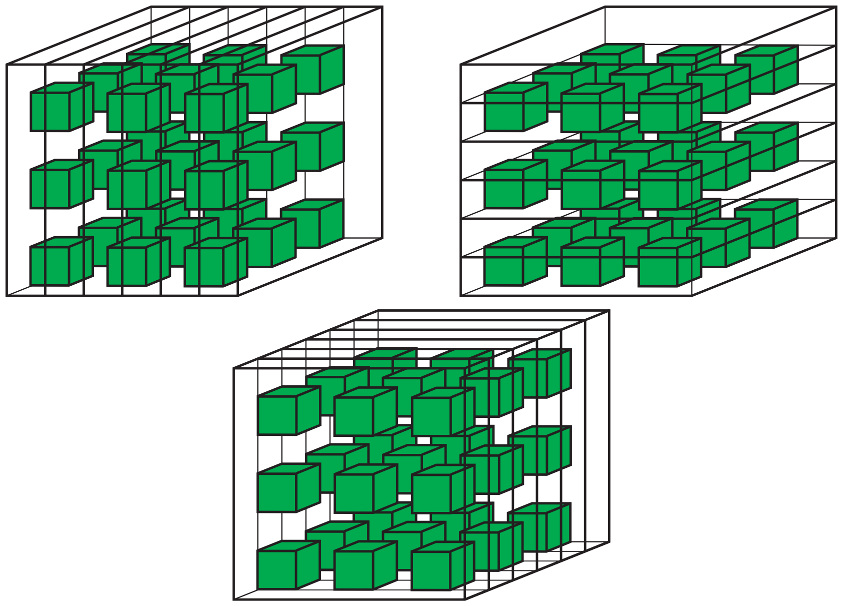


FIG. 7. *Three different* 3*-dimensional slices of the subdivision along three hyperplanes orthogonal to* $x_4$.

Since all the rods have pairwise disjoint interiors, we can obtain an orthogonal subdivision of the container box $R = [0, 2\ell] \times [0, 2\ell] \times [0, 2\ell] \times [0, 3\ell^{3/2}]$ by filling up the space between the rods with $9\ell^{9/2}$ unit cubes. This completes the description of our auxiliary construction of $\Theta(\ell^{9/2})$ boxes. We obtain a subdivision $S$ of $n = \Theta(\ell^{9/2})$ boxes in the following way: Partition each 2-rod into $\ell^2$ congruent 2-rods with the same orientation, and replace each 1-rod by $\ell^{3/2}$ pairwise disjoint 1-rods whose projection to the $(x_2, x_3, x_4)$ hyperplane is the worst-case construction for a subdivision in $\mathbb{R}^3$.

Consider the $\ell^{9/2}$ boxes $\Pi(i, j, k, t) = [2i, 2i+2] \times [2j, 2j+2] \times [2k, 2k+2] \times [3t, 3t+3]$, where $i, j, k = 0, 1, \ldots, \ell - 1$ and $t = 0, 1, \ldots, \ell^{3/2} - 1$. These are the *junction* regions. Note that the junctions have pairwise disjoint interiors. We show that any axis-aligned BSP for $S$ dissects the rods of $S$ such that pieces of rods have at least $\ell^2$ vertices in the interior of every junction. This implies that the total number of pieces is $\Omega(\ell^{9/2} \cdot \ell^2) = \Omega(\ell^{13/2}) = \Omega(n^{13/9})$.

First, suppose that each of the $\ell^{3/2}$ 1-rods in $r_1(i, j, k) \in R_1$ is cut by some hyperplane $x_4 = c$ for $c \in (3t, 3t + 3)$. (Not all 1-rods may be cut by the same hyperplane, though.) Consider an intersection of the bundle of 1-rods with a hyperplane $x_4 = c_0$ such that each 1-rod has a vertex in the interior of $\Pi(i, j, k, t)$ below $x_4 = c_0$. A BSP for $S$, restricted to the intersection of the bundle with $x_4 = c_0$, must be a BSP for the 3-dimensional subdivision to which the bundle projects. Hence the intersection BSP must have $\Omega((\ell^{3/2})^{4/3}) = \Omega(\ell^2)$ vertices, by the previous lower bound. Each vertex corresponds to a vertex in the interior of $\Pi(i, j, k, t)$—in fact, it is connected to its corresponding vertex by an edge of the BSP parallel to the $x_4$-axis.

Otherwise, let $r$ be a rod in $R_1(i, j, k)$ that is not cut by any hyperplane $x_4 = c$, for $c \in (3t, 3t + 3)$. Let $e$ be a segment in the interior of $r$ such that $e$ is pass-through in $x_4$ within $\Pi(i, j, k, t)$ and does not lie on any hyperplane of the BSP. Since $e$ is not cut by the BSP, one subproblem contains $e$ all through the partitioning procedure. Notice that the container box for $e$ cannot be separated from all the other original boxes by hyperplanes orthogonal to only one axis. Let $H_1$ and $H_2$ be hyperplanes orthogonal to, say, $x_1$ and $x_2$, on the boundary of the final container box of $e$. The 2-face $F = H_1 \cap H_2$ is pass-through in $x_3$ and $x_4$ and lies in the interior of $\Pi(i, j, k, t)$. Therefore $F$ has a common vertex with all $\ell^2$ 2-rods in $r_{1,2}(k, t)$ in the interior of $\Pi(i, j, k, t)$. ◻

Generalizing this recursive construction, we obtain a recursive formula for the exponent of the lower bound $\beta(d)$ we can reach for a box subdivision in $\mathbb{R}^d$.

THEOREM 6.3. *There is a $d$-dimensional box subdivision $S$ with $n$ boxes such that the size of any axis-aligned BSP for $S$ is $\Omega(n^{\beta(d)})$, where $\beta(3) = 4/3$, $\beta(4) = 13/9$, and $\beta(d)$ converges to $(1 + \sqrt{5})/2$ as $d \to \infty$.*

*Proof.* In $\mathbb{R}^d$, our auxiliary construction consists of $\ell^{d-1}$ 1-rods that are pass-through in $x_d$, and $(d-1)$ families of $\ell^{1+\alpha(d)}$ $(d-2)$-rods such that each slice $x_d = c$ intersects $\ell$ $(d-2)$-rods and there are $\ell^{\alpha(d)}$ levels in the $x_d$ extent ($\alpha(d)$ is a parameter to be specified later). We need $\Theta(\ell^{(d-1)+\alpha(d)})$ unit cubes to convert this configuration into a subdivision.

We balance the number of filler unit cubes and rods by replacing each auxiliary $(d-2)$-rod with a bundle of $\ell^{d-2}$ congruent $(d-2)$-rods and each auxiliary 1-rod by a bundle of $\ell^{\alpha(d)}$ 1-rods. We arrange the 1-rods in each bundle such that their $(d-1)$-dimensional projection gives the $\mathbb{R}^{d-1}$ subdivision for which any BSP has size $\Omega((\ell^{\alpha(d)})^{\beta(d-1)})$.

We define $\ell^{(d-1)+\alpha(d)}$ junctions, arranged in a grid. Each junction contains either $\Omega(\ell^{d-2})$ vertices on fragments of $(d-2)$-rods or $\Omega((\ell^{\alpha(d)})^{\beta(d-1)})$ vertices on fragments of 1-rods. To achieve our bound, we set $\alpha(d) = (d-2)/\beta(d-1)$, implying that any BSP for our subdivision has at least $\ell^{(d-1)+\alpha(d)} \cdot \ell^{d-2} = \ell^{2d-3+\alpha(d)}$ vertices. That gives a lower bound of

$$\Omega\left(n^{\frac{2d-3+\alpha(d)}{d-1+\alpha(d)}}\right) = \Omega\left(n^{\frac{2d-3+(d-2)/\beta(d-1)}{d-1+(d-2)/\beta(d-1)}}\right).$$

The exponent in this bound is

$$\beta(d) = \frac{2d-3+(d-2)/\beta(d-1)}{d-1+(d-2)/\beta(d-1)} = 1 + \frac{d-2}{d-1+(d-2)/\beta(d-1)}.$$

The first values of this sequence are $\beta(4) = 13/9 = 1.444$, $\beta(5) = 118/79 = 1.494$, $\beta(6) = 689/453 = 1.521$, and $\beta(7) = 9844/6399 = 1.538$. The sequence converges to $\lim_{d\to\infty} \beta(d) = (1 + \sqrt{5})/2 = 1.618$.

Note that the $d$-dimensional lower bound subsumes the lower bound for three dimensions. If $d = 3$, then $\beta(d-1) = \beta(2) = 1$, and $\alpha(3) = (3-2)/\beta(2) = 1$. The lower bound we obtain is

$$\Omega\left(n^{\frac{2\cdot 3-3+\alpha(3)}{3-1+\alpha(3)}}\right) = \Omega(n^{4/3}),$$

matching Theorem 6.1. ◻

## 7. Concluding remarks.

The BSP is a popular space decomposition method in computational geometry, computer graphics, and other fields dealing with geometric shapes. However, its worst-case complexity remains poorly understood in dimensions three and higher. Even for the BSP of $n$ orthogonal boxes in $d$ dimensions, the best upper bound known is $O(n^{d/2})$, while no lower bound better than $\Omega(n^2)$ is known.

In an attempt to understand the complexity of higher-dimensional BSPs, we considered a natural special case: orthogonal subdivisions. We showed that every

$d$-dimensional subdivision with $n$ boxes admits a BSP of size $O(n^{(d+1)/3})$. We also exhibited subdivisions in $d$ dimensions for which every axis-aligned BSP must have size $\Omega(n^{\beta(d)})$, where $\beta(d)$ converges to $(1+\sqrt{5})/2$ as $d \to \infty$. Our result shows that if the objects do not fracture the complement space too badly, then their BSP size can be significantly smaller than the current worst-case bounds indicate.

Clearly, the most interesting open problem suggested by this paper is to close the gap between the upper and lower bounds. The absence of superquadratic lower bounds is intriguing. Can it really be the case that, in any dimension $d$, a set of $n$ boxes (forming a subdivision, or not) admits a quadratic size BSP? If not, then what is the true complexity of the BSP of $n$ $d$-dimensional boxes? Is this bound necessarily worse than that for $n$ boxes forming a subdivision in higher dimensions?

## REFERENCES

[1] P. K. Agarwal, E. F. Grove, T. M. Murali, and J. S. Vitter, *Binary space partitions for fat rectangles*, SIAM J. Comput., 29 (2000), pp. 1422–1448.

[2] C. Ballieux, *Motion Planning Using Binary Space Partitions*, Tech. report Inf/src/93-25, Utrecht University, Utrecht, The Netherlands, 1993.

[3] J. L. Bentley, *Multidimensional binary search trees used for associative searching*, Comm. ACM, 18 (1975), pp. 509–517.

[4] M. de Berg, *Linear size binary space partitions for uncluttered scenes*, Algorithmica, 28 (2000), pp. 353–366.

[5] P. Berman, B. DasGupta, and S. Muthukrishnan, *Exact size of binary space partitionings and improved rectangle tiling algorithms*, SIAM J. Discrete Math., 15 (2002), pp. 252–267.

[6] N. Chin and S. Feiner, *Near real-time shadow generation using BSP trees*, in Proceedings of the 16th Annual Conference on Computer Graphics and Interactive Techniques, ACM, New York, 1989, pp. 99–106.

[7] N. Chin and S. Feiner, *Fast object-precision shadow generation for area light sources using BSP trees*, in Proceedings of the 1992 Symposium on Interactive 3D Graphics, ACM, New York, 1992, pp. 21–30.

[8] A. Dumitrescu, J. S. B. Mitchell, and M. Sharir, *Binary space partitions for axis-parallel segments, rectangles, and hyperrectangles*, Discrete Comput. Geom., 31 (2004), pp. 207–227.

[9] D. Eppstein, *Three Untetrahedralizable Objects*, The Geometry Junkyard, available online at http://www.ics.uci.edu/~eppstein/junkyard/untetra.

[10] H. Fuchs, Z. M. Kedem, and B. Naylor, *On visible surface generation by a priori tree structures*, in Proceedings of the 7th Annual Conference on Computer Graphics and Interactive Techniques, ACM, New York, 1980, pp. 124–133.

[11] T. M. Murali, *Efficient Hidden-Surface Removal in Theory and in Practice*, Ph.D. thesis, Department of Computer Science, Brown University, Providence, RI, 1998, pp. 80–81.

[12] B. Naylor, J. A. Amanatides, and W. Thibault, *Merging BSP trees yields polyhedral set operations*, in Proceedings of the 17th Annual Conference on Computer Graphics and Interactive Techniques, ACM, New York, 1990, pp. 115–124.

[13] B. Naylor and W. Thibault, *Application of BSP Trees to Ray-Tracing and CSG Evaluation*, Tech. report GIT-ICS 86/03, School of Information and Computer Science, Georgia Institute of Technology, Atlanta, GA, 1986.

[14] M. S. Paterson and F. F. Yao, *Efficient binary space partitions for hidden-surface removal and solid modeling*, Discrete Comput. Geom., 5 (1990), pp. 485–503.

[15] M. S. Paterson and F. F. Yao, *Optimal binary space partitions for orthogonal objects*, J. Algorithms, 13 (1992), pp. 99–113.

[16] R. A. Schumacker, R. Brand, M. Gilliland, and W. Sharp, *Study for Applying Computer-Generated Images to Visual Simulation*, Tech. report AFHRL–TR–69–14, U.S. Air Force Human Resources Laboratory, San Antonio, TX, 1969.

[17] W. C. Thibault and B. F. Naylor, *Set operations on polyhedra using binary space partitioning trees*, in Proceedings of the 14th Annual Conference on Computer Graphics and Interactive Techniques, ACM, New York, 1987, pp. 153–162.

[18] C. D. Tóth, *A note on binary plane partitions*, Discrete Comput. Geom., 30 (2003), pp. 3–16.

[19] C. D. Tóth, *Binary space partitions for orthogonal fat rectangles*, in Proceedings of the 11th European Symposium on Algorithms, Lecture Notes in Comput. Sci. 2832, Springer-Verlag, Berlin, 2003, pp. 494–505.

SIAM J. COMPUT.
Vol. 34, No. 6, pp. 1398–1431

# A SHORTEST PATH ALGORITHM FOR REAL-WEIGHTED UNDIRECTED GRAPHS[*]

SETH PETTIE[†] AND VIJAYA RAMACHANDRAN[‡]

**Abstract.** We present a new scheme for computing shortest paths on real-weighted undirected graphs in the fundamental *comparison-addition model*. In an efficient preprocessing phase our algorithm creates a linear-size structure that facilitates single-source shortest path computations in $O(m \log \alpha)$ time, where $\alpha = \alpha(m, n)$ is the very slowly growing inverse-Ackermann function, $m$ the number of edges, and $n$ the number of vertices. As special cases our algorithm implies new bounds on both the all-pairs and single-source shortest paths problems. We solve the all-pairs problem in $O(mn \log \alpha(m, n))$ time and, if the ratio between the maximum and minimum edge lengths is bounded by $n^{(\log n)^{O(1)}}$, we can solve the single-source problem in $O(m + n \log \log n)$ time. Both these results are theoretical improvements over Dijkstra's algorithm, which was the previous best for real weighted undirected graphs. Our algorithm takes the hierarchy-based approach invented by Thorup.

**Key words.** single-source shortest paths, all-pairs shortest paths, undirected graphs, Dijkstra's algorithm

**AMS subject classifications.** 05C12, 05C85, 68R10

**DOI.** 10.1137/S0097539702419650

## 1. Introduction.

The problem of computing shortest paths is indisputably one of the most well-studied problems in computer science. It is thoroughly surprising that in the setting of *real*-weighted graphs, many basic shortest path problems have seen little or no progress since the early work by Dijkstra, Bellman and Ford, Floyd and Warshall, and others [CLRS01]. For instance, no algorithm for computing single-source shortest paths (SSSPs) in arbitrarily weighted graphs has yet to improve the Bellman–Ford $O(mn)$ time bound, where $m$ and $n$ are the number of edges and vertices, respectively. The fastest uniform all-pairs shortest path (APSP) algorithm for dense graphs [Z04, F76] requires time $O(n^3\sqrt{\log \log n}/\log n)$, which is just a slight improvement over the $O(n^3)$ bound of the Floyd–Warshall algorithm. Similarly, Dijkstra's $O(m + n \log n)$ time algorithm [Dij59, FT87] remains the best for computing SSSPs on nonnegatively weighted graphs, and until the recent algorithms of Pettie [Pet04, Pet02b, Pet03], Dijkstra's algorithm was also the best for computing APSPs on sparse graphs [Dij59, J77, FT87].

In order to improve these bounds most shortest path algorithms depend on a restricted type of input. There are algorithms for geometric inputs (see Mitchell's survey [Mit00]), planar graphs [F91, HKRS97, FR01], and graphs with randomly chosen edge weights [Spi73, FG85, MT87, KKP93, KS98, M01, G01, Hag04]. In recent years there

[*]Received by the editors December 13, 2002; accepted for publication (in revised form) October 15, 2004; published electronically July 26, 2005. This work was supported by Texas Advanced Research Program grant 003658-0029-1999 and NSF grant CCR-9988160. A preliminary version of this paper, titled *Computing shortest paths with comparisons and additions*, was presented at the 13th Annual ACM-SIAM Symposium on Discrete Algorithms, 2002, San Francisco, CA.

http://www.siam.org/journals/sicomp/34-6/41965.html

[†]Max Planck Institut für Informatik, Stuhlsatzenhausweg 85, 66123 Saarbrücken, Germany (pettie@mpi-sb.mpg.de). This author's work was also supported by an Alexander von Humboldt Postdoctoral Fellowship and by an MCD Graduate Fellowship.

[‡]Department of Computer Sciences, The University of Texas at Austin, Austin, TX 78712 (vlr@cs.utexas.edu).

has also been a focus on computing approximate shortest paths—see Zwick's recent survey [Z01]. One common assumption is that the graph is *integer*-weighted, though structurally unrestricted, and that the machine model is able to manipulate the integer representation of weights. Shortest path algorithms based on scaling [G85b, GT89, G95] and fast matrix multiplication [Sei95, GM97, AGM97, Tak98, SZ99, Z02] have running times that depend on the magnitude of the integer edge weights, and therefore yield improved algorithms only for sufficiently small edge weights. In the case of the matrix multiplication–based algorithms the critical threshold is rather low: even edge weights sublinear in $n$ can be too large. Dijkstra's algorithm can be sped up in the integer-weight model by using an integer priority queue.[1] The best bounds on Dijkstra's algorithm to date are $O(m\sqrt{\log\log n})$ (expected) [HT02] and $O(m + n\log\log n)$ [Tho03]. Both of these algorithms use multiplication, a non-$AC^0$ operation; see [Tho03] for bounds in the $AC^0$ model. Thorup [Tho99] considered the restricted case of integer-weighted undirected graphs and showed that on an $AC^0$ random access machine (RAM), shortest paths could be computed in linear time. Thorup invented what we call in this paper the *hierarchy-based* approach to shortest paths.

The techniques developed for integer-weighted graphs (scaling, matrix multiplication, integer sorting, and Thorup's hierarchy-based approach) *seem* to depend crucially on the graph being integer-weighted. This state of affairs is not unique to the shortest path problem. In the weighted matching [G85b, GT89, GT91] and maximum flow problems [GR98], for instance, the best algorithms for real- and integer-weighted graphs have running times differing by a *polynomial* factor. For the shortest path problem on positively weighted graphs the integer/real gap is only logarithmic. It is of great interest whether an integer-based approach is inherently so, or whether it can yield a faster algorithm for general, real-weighted inputs. In this paper we generalize Thorup's hierarchy-based approach to the comparison-addition model (see section 2.1) and, as a corollary, to real-weighted input graphs. For the undirected APSP problem we nearly eliminate the existing integer/real gap, reducing it from $\log n$ to $\log\alpha(m,n)$, where $\alpha$ is the incomprehensibly slowly growing inverse-Ackermann function. Before stating our results in detail, we first give an overview of the hierarchy-based approach and discuss the recent hierarchy-based shortest path algorithms [Tho99, Hag00, Pet04, Pet02b].

Hierarchy-based algorithms should be thought of as preprocessing schemes for answering SSSP queries in nonnegatively weighted graphs. The idea is to compute a small non–source-specific structure that encodes useful information about all the shortest paths in the graph. We measure the running time of a hierarchy-based algorithm with two quantities: $\mathcal{P}$, the worst case preprocessing cost on the given graph, and $\mathcal{M}$, the marginal cost of one SSSP computation after preprocessing. Therefore, solving the $s$-sources shortest path problem requires $s\cdot\mathcal{M}+\mathcal{P}$ time. If $s = n$, that is, if we are solving APSP, then for all known hierarchy algorithms the $\mathcal{P}$ term becomes negligible. However, $\mathcal{P}$ may be dominant (in either the asymptotic or real-world sense) for smaller values of $s$. In Thorup's original algorithm [Tho99], $\mathcal{P}$ and $\mathcal{M}$ are both $O(m)$; recall that his algorithm works on integer-weighted undirected graphs. Hagerup [Hag00] adapted Thorup's algorithm to integer-weighted *directed* graphs, incurring a slight loss of efficiency in the process. In [Hag00], $\mathcal{P} = O(\min\{m\log\log C, m\log n\})$,[2]

[1]It can also be sped up using an integer sorting algorithm in conjunction with Thorup's reduction [Tho00] from priority queues to sorting.

[2]Hagerup actually proved $\mathcal{P} = O(\min\{m\log\log C, mn\})$; see [Pet04] for the $O(m\log n)$ bound.

where $C$ is the maximum edge weight and $\mathcal{M} = O(m + n \log\log n)$. After the initial publication of our results [PR02a], Pettie [Pet04, Pet02b] gave an adaptation of the hierarchy-based approach to *real*-weighted directed graphs. The main result of [Pet04] is an APSP algorithm running in time $O(mn + n^2 \log\log n)$, which improved upon the $O(mn + n^2 \log n)$ bound derived from multiple runs of Dijkstra's algorithm [Dij59, J77, FT87]. The result of [Pet04] is stated in terms of the APSP problem because its preprocessing cost $\mathcal{P}$ is $O(mn)$, making it efficient only if $s$ is very close to $n$. In [Pet02b] (see also [Pet03]) the *nonuniform* complexity of APSP is considered; the main result of [Pet02b] is an algorithm performing $O(mn \log \alpha(m, n))$ comparison and addition operations. This bound is essentially optimal when $m = O(n)$ due to the trivial $\Omega(n^2)$ lower bound on APSP.

In this paper we give new bounds on computing *undirected* shortest paths in real-weighted graphs. For our algorithm, the preprocessing cost $\mathcal{P}$ is $O(\textsc{mst}(m, n) + \min\{n \log n, n \log\log r\})$, where $\textsc{mst}(m, n)$ is the complexity of the minimum spanning tree problem and $r$ is the ratio of the maximum-to-minimum edge weight. This bound on $\mathcal{P}$ is never worse than $O(m + n \log n)$, though if $r$ is not excessively large, say less than $n^{(\log n)^{O(1)}}$, $\mathcal{P}$ is $O(m + n \log\log n)$. We show that the marginal cost $\mathcal{M}$ of our algorithm is asymptotically equivalent to $\textsc{split-findmin}(m, n)$, which is the decision-tree complexity of a certain data structuring problem of the same name. It was known that $\textsc{split-findmin}(m, n) = O(m\alpha(m, n))$ [G85a]; we improve this bound to $O(m \log \alpha(m, n))$. Therefore, the marginal cost of our algorithm is essentially (but perhaps not precisely) linear. Theorem 1.1 gives our general result, and Corollaries 1.2 and 1.3 relate it to the canonical APSP and SSSP problems, respectively.

THEOREM 1.1. *Let $\mathcal{P} = \textsc{mst}(m, n) + \min\{n \log n, n \log\log r\}$, where $m$ and $n$ are the number of edges and vertices in a given undirected graph, $r$ bounds the ratio of any two edge lengths, and $\textsc{mst}(m, n)$ is the cost of computing the graph's minimum spanning tree. In $O(\mathcal{P})$ time an $O(n)$-space structure can be built that allows the computation of SSSPs in $O(\textsc{split-findmin}(m, n))$ time, where $\textsc{split-findmin}(m, n) = O(m \log \alpha(m, n))$ represents the decision-tree complexity of the split-findmin problem and $\alpha$ is the inverse-Ackermann function.*

COROLLARY 1.2. *The undirected APSP problem can be solved on a real-weighted graph in $O(n \cdot \textsc{split-findmin}(m, n)) = O(mn \log \alpha(m, n))$ time.*

COROLLARY 1.3. *The undirected SSSP problem can be solved on a real-weighted graph in $O(\textsc{split-findmin}(m, n) + \textsc{mst}(m, n) + \min\{n \log n, n \log\log r\}) = O(m\alpha(m, n) + \min\{n \log n, n \log\log r\})$ time.*

The running time of our SSSP algorithm (Corollary 1.3) is rather unusual. It consists of three terms, where the first two are unknown (but bounded by $O(m\alpha(m, n))$) and the third depends on a nonstandard parameter: the maximum ratio of any two edge lengths.[3] A natural question is whether our SSSP algorithm can be substantially improved. In section 6 we formally define the class of "hierarchy-based" SSSP algorithms and show that any comparison-based undirected SSSP algorithm in this class must take time $\Omega(m + \min\{n \log n, n \log\log r\})$. This implies that our SSSP algorithm is optimal for this class, up to an inverse-Ackermann factor, and that no hierarchy-based SSSP algorithm can improve on Dijkstra's algorithm, for $r$ unbounded.

Pettie, Ramachandran, and Sridhar [PRS02] implemented a simplified version of our algorithm. The observed marginal cost of the [PRS02] implementation is nearly linear, which is in line with our asymptotic analysis. Although it is a little slower

[3]Dinic's implementation [Din78, Din03] of Dijkstra's algorithm also depends on $r$, in both time and space consumption.

than Dijkstra's algorithm in solving SSSP, it is faster in solving the $s$-sources shortest path problem, in some cases for $s$ as small as 3. In many practical situations it is the $s$-sources problem, not SSSP, that needs to be solved. For instance, if the graph represents a physical network, such as a network of roads or computers, it is unlikely to change very often. Therefore, in these situations a nearly linear preprocessing cost is a small price to pay for more efficient shortest path computations.

**1.1. An overview.** In section 2 we define the SSSP and APSP problems and review the comparison-addition model and Dijkstra's algorithm [Dij59]. In section 3 we generalize the hierarchy approach to real-weighted graphs and give a simple proof of its correctness. In section 4 we propose two implementations of the general hierarchy-based algorithm, one for proving the asymptotic bounds of Theorem 1.1 and one that is simpler and uses more standard data structures. The running times of our implementations depend heavily on having a *well-balanced* hierarchy. In section 5 we give an efficient method for constructing balanced hierarchies; it is based on a hierarchical clustering of the graph's minimum spanning tree. In section 6 we prove a lower bound on the class of hierarchy-based undirected SSSP algorithms. In section 7 we discuss avenues for further research.

## 2. Preliminaries.

The input is a weighted, undirected graph $G = (V, E, \ell)$, where $V = V(G)$ and $E = E(G)$ are the sets of $n$ vertices and $m$ edges, respectively, and $\ell : E \to \mathbb{R}$ assigns a real length to each edge. The *distance* from vertex $u$ to vertex $v$, denoted $d(u, v)$, is the length of the minimum length path from $u$ to $v$, or $\infty$ if there is no such path from $u$ to $v$, or $-\infty$ if there is no such path of minimum length. The APSP problem is to compute $d(u, v)$, for $(u, v) \in V \times V$, and the SSSP problem is to compute $d(u, v)$ for some specified *source* $u$ and all $v \in V$.

If, in an undirected graph, some connected component contains an edge of negative length, say $e$, then the distance between two vertices $u$ and $v$ in that component is $-\infty$: one can always construct a path of arbitrarily small length by concatenating a path from $u$ to $e$, followed by the repetition of $e$ a sufficient number of times, followed by a path from $e$ to $v$. Without loss of generality we will assume that $\ell : E \to \mathbb{R}^+$ assigns only positive edge lengths. A slightly restricted problem (which forbids the types of paths described above) is the shortest *simple* path problem. This problem is NP-hard as it generalizes the Hamiltonian path problem. However, Edmonds showed that when there is no negative weight simple cycle, the problem is solvable in polynomial time by a reduction to weighted matching—see [AMO93, p. 496] and [G85a].

### 2.1. The comparison-addition model.

We use the term *comparison-addition model* to mean any uniform model in which real numbers are subject to only comparison and addition operations. The term *comparison-addition complexity* refers to the number of comparison and addition operations, ignoring other computational costs. In the comparison-addition model we leave unspecified the machine model used for all data structuring tasks. Our results as stated hold when that machine model is a RAM. If instead we assume a pointer machine [Tar79], our algorithms slow down by at most an inverse-Ackermann factor.[4]

The comparison-addition model has some aesthetic appeal because it is the simplest model appropriate to computing shortest paths and many other network opti-

[4]The only structure we use whose complexity changes between the RAM and pointer machine models is the split-findmin structure. On a pointer machine there are matching upper and lower bounds of $\Theta(m\alpha)$ [G85a, LaP96], whereas on the RAM the complexity is somewhere between $\Omega(m)$ and $O(m \log \alpha)$—see Appendix B.

mization problems. A common belief is that simplicity is necessarily gained at the price of practicality; however, this is not true. In the setting of an algorithms library, such as LEDA [MN00], it is important—and practical—that *data types* be fully separated from *algorithms* and that the interface between the two be as generic as possible. There is always room for fast algorithms specialized to integers or floats. However, even under these assumptions, the gains in speed can be surprisingly minor; see [PRS02] for one example.

**2.1.1. Techniques.** In our algorithm we sometimes use subtraction on real numbers, an operation that is not directly available in the comparison-addition model. Lemma 2.1, given below, shows that simulating subtraction incurs at most a constant factor loss in efficiency.

LEMMA 2.1. *$C$ comparisons and $A$ additions and subtractions can be simulated in the comparison-addition model with $C$ comparisons and $2(A+C)$ additions.*

*Proof.* We represent each real $x_i = a_i - b_i$ as two reals $a_i, b_i$. An addition $x_i + x_j = (a_i + a_j) - (b_i + b_j)$ or a subtraction $x_i - x_j = (a_i + b_j) - (b_i + a_j)$ can be simulated with two actual additions. A comparison $x_i : x_j$ is equivalent to the comparison $a_i + b_j : a_j + b_i$, which involves two actual additions and a comparison. □

At a key point in our algorithm we need to approximate the ratio of two numbers. Division is clearly not available for real numbers in the comparison-addition model, and with a little thought one can see that it cannot be simulated exactly. Lemma 2.2, given below, bounds the time to find certain *approximate* ratios in the comparison-addition model, which will be sufficient for our purposes.

LEMMA 2.2. *Let $p_1, \dots, p_k$ be real numbers, where $p_1$ and $p_k$ are the smallest and largest, respectively. We can find the set of integers $\{q_i\}$ such that $2^{q_i} \le \frac{p_i}{p_1} < 2^{q_i+1}$ in $\Theta(\log \frac{p_k}{p_1} + k \log\log \frac{p_k}{p_1})$ time.*

*Proof.* We generate the set $\mathcal{L} = \{p_1, 2 \cdot p_1, 4 \cdot p_1, \dots, 2^{\lceil \log \frac{p_k}{p_1} \rceil} \cdot p_1\}$ with $\log \frac{p_k}{p_1}$ additions; then for each $p_i$ we find $q_i$ in $\log |\mathcal{L}| = O(\log\log \frac{p_k}{p_1})$ time with a binary search over $\mathcal{L}$. □

In our algorithm the $\{p_i\}$ correspond to certain edge lengths, and $k = \Theta(n)$. Our need to approximate ratios, as in Lemma 2.2, is the source of the peculiar $n \log\log r$ term in the running time of Theorem 1.1. We note here that the bound stated in Lemma 2.2 is pessimistic in the following sense. If we randomly select the $\{p_i\}$ from a uniform distribution (or other natural distribution), then the time to find approximate ratios can be reduced to $O(k)$ (with high probability) using a linear search rather than a binary search.

**2.1.2. Lower bounds.** There are many lower bounds for shortest path problems in the comparison-addition model, though none are truly startling. Spira and Pan [SP75] showed that even if additions are free, $\Omega(n^2)$ comparisons are necessary to solve SSSP on the complete graph. Karger, Koller, and Phillips [KKP93] proved that directed APSP requires $\Omega(mn)$ comparisons if each summation corresponds to a path in the graph.[5] Kerr [K70] showed that any oblivious APSP algorithm performs $\Omega(n^3)$ comparisons, and Kolliopoulos and Stein [KS98] proved that any fixed sequence of edge relaxations solving SSSP must have length $\Omega(mn)$. By "fixed sequence" they mean one that depends only on $m$ and $n$ but not on the graph structure. Ahuja et al. [AMOT90] observed that any implementation of Dijkstra's algorithm requires $\Omega(m + n \log n)$ comparison and addition operations. Pettie [Pet04] gave an $\Omega(m +$

[5]However, it is not true that all shortest path algorithms satisfy this condition. For example, our algorithm does not, and neither do [F76, Tak92, Han04, Z04, Pet04, Pet02b].

$\min\{n \log r, n \log n\})$ lower bound on computing directed SSSP with a "hierarchy-type" algorithm, where $r$ bounds the ratio of any two edge lengths. In section 6 we prove a lower bound of $\Omega(m + \min\{n \log\log r, n \log n\})$ on hierarchy-type algorithms for *undirected* SSSP. These last two lower bounds are essentially tight for hierarchy-type algorithms, on directed and undirected graphs, respectively.

Graham, Yao, and Yao [GYY80] proved that the information-theoretic argument cannot prove a nontrivial $\omega(n^2)$ lower bound on the *comparison*-complexity of APSP, where additions are granted for free. It is also simple to see that there can be no nontrivial information-theoretic lower bound on SSSP.

**2.2. Dijkstra's algorithm.** Our algorithm, like [Tho99, Hag00], is best understood as circumventing the limitations of Dijkstra's algorithm. We give a brief description of Dijkstra's algorithm in order to illustrate its complexity and introduce some vocabulary.

For a vertex set $S \subseteq V(G)$, let $d_S(u, v)$ denote the distance from $u$ to $v$ in the subgraph induced by $S \cup \{v\}$. Dijkstra's algorithm maintains a *tentative* distance function $D(v)$ and a set of *visited* vertices $S$ satisfying Invariant 2.1. Henceforth, $s$ denotes the source vertex.

INVARIANT 2.1. *Let $s$ be the source vertex and $v$ be an arbitrary vertex:*

$$D(v) = \begin{cases} d(s, v) & \text{if } v \in S, \\ d_S(s, v) & \text{if } v \notin S. \end{cases}$$

Choosing an initial assignment of $S = \emptyset$, $D(s) = 0$, and $D(v) = \infty$ for $v \neq s$ clearly satisfies the invariant. Dijkstra's algorithm consists of repeating the following step $n$ times: choose a vertex $v \in V(G) \backslash S$ such that $D(v)$ is minimized, set $S := S \cup \{v\}$, and finally, update tentative distances to restore Invariant 2.1. This last part involves *relaxing* each edge $(v, w)$ by setting $D(w) = \min\{D(w), D(v) + \ell(v, w)\}$. Invariant 2.1 and the positive-weight assumption imply $D(v) = d(s, v)$ when $v$ is selected. It is also simple to prove that relaxing outgoing edges of $v$ restores Invariant 2.1.

The problem with Dijkstra's algorithm is that vertices are selected in increasing distance from the source, a task that is at least as hard as sorting $n$ numbers. Maintaining Invariant 2.1, however, does not demand such a particular ordering. In fact, it can be seen that selecting *any* vertex $v \notin S$ for which $D(v) = d(s, v)$ will maintain Invariant 2.1. All hierarchy-type algorithms [Tho99, Hag00, Pet04, Pet02b] maintain Invariant 2.1 by generating a weaker certificate for $D(v) = d(s, v)$ than "$D(v)$ is minimal." Any such certificate must show that for all $u \notin S$, $D(u) + d(u, v) \geq D(v)$. For example, Dijkstra's algorithm presumes there are no negative length edges, hence $d(u, v) \geq 0$, and by choice of $v$ ensures $D(u) \geq D(v)$. This is clearly a sufficient certificate. In Dinic's version [Din78] of Dijkstra's algorithm the lower bound $d(u, v) \geq \ell_{\min}$ is used, where $\ell_{\min}$ is the minimum edge length. Thus Dinic is free to visit any $v \notin S$ for which $\lfloor D(v)/\ell_{\min} \rfloor$ is minimal. All hierarchy-type algorithms [Tho99, Hag00, Pet04, Pet02b], ours included, precompute a much stronger lower bound on $d(u, v)$ than $d(u, v) \geq \ell_{\min}$.

## 3. The hierarchy approach and its correctness.

In this section we generalize the hierarchy-based approach of [Tho99] to real-weighted graphs. Because the algorithm follows directly from its proof of correctness, we will actually give a kind of correctness proof first.

Below, $X \subseteq V(G)$ denotes any set of vertices, and $s$ always denotes the source vertex. Let $I$ be a real interval. The notation $X^I$ refers to the subset of $X$ whose distance from the source lies in the interval $I$, i.e.,

$$X^I = \{ v \in X \ : \ d(s,v) \in I \}.$$

DEFINITION 3.1. *A vertex set $X$ is $(S, [a,b))$-SAFE if* (i) $X^{[0,a)} \subseteq S$,
(ii) *for* $v \in X^{[a,b)}$, $d_{S \cup X}(s,v) = d(s,v)$.

In other words, if a subgraph is $(S, I)$-SAFE, we can determine the distances that lie in interval $I$ without looking at parts of the graph outside the subgraph and $S$. Clearly, finding safe subgraphs has the potential to let us compute distances cheaply.

DEFINITION 3.2. *A set $\{X_i\}_i$ is a $t$-partition of $X$ if the $\{X_i\}_i$ partition $X$ and for every edge $(u,v)$ with $u \in X_i$, $v \in X_j$, and $i \neq j$, we have $\ell(u,v) \geq t$.*

Note that a $t$-partition need not be maximal; that is, if $\{X_1, X_2, \ldots, X_k\}$ is a $t$-partition, then $\{X_1 \cup X_2, X_3, \ldots, X_k\}$ is as well.

LEMMA 3.3. *Suppose that $X$ is $(S, [a,b))$-SAFE. Let $\{X_i\}_i$ be a $t$-partition of $X$ and let $S'$ be such that $S \cup X^{[a, \min\{a+t,b\})} \subseteq S'$. Then*
(i) *for $X_i$ in the $t$-partition, $X_i$ is $(S, [a, \min\{a+t, b\}))$-SAFE;*
(ii) *$X$ is $(S', [\min\{a+t,b\}, b))$-SAFE.*

*Proof.* We prove part (i) first. Let $v \in X_i^{[a,\min\{a+t,b\})}$ and suppose that the lemma is false, that $d(s,v) \neq d_{S \cup X_i}(s,v)$. From the assumed safeness of $X$ we know that $d(s,v) = d_{S \cup X}(s,v)$. This means that the shortest path to $v$ must pass through $X \backslash (X_i \cup S)$. Let $w$ be the last vertex in $X \backslash (X_i \cup S)$ on the shortest $s$–$v$ path. By Definition 3.2, the edge from $w$ to $X_i$ has length $\geq t$. Since $d(s,v) < \min\{a+t, b\}$, $d(s,w) < \min\{a+t,b\} - t \leq a$. Since, by Definition 3.1(i), $X^{[0,a)} \in S$, it must be that $w \in S$, contradicting our selection of $w$ from $X \backslash (X_i \cup S)$. Part (ii) claims that $X$ is $(S', [\min\{a+t,b\}, b))$-SAFE. Consider first Definition 3.1(i) regarding safeness. By the assumption that $X$ is $(S, [a,b))$-SAFE we have $X^{[0,a)} \subseteq S$, and by definition of $S'$ we have $S \cup X^{[a,\min\{a+t,b\})} \subseteq S'$; therefore $X^{[0,\min\{a+t,b\})} \subseteq S'$, satisfying Definition 3.1(i). By the assumption that $X$ is $(S,[a,b))$-SAFE we have that for $v \in X^{[a,b)}$, $d_{S\cup X}(s,v) = d(s,v)$; this implies the weaker statement that for $v \in X^{[\min\{a+t,b\},b)}$, $d_{S' \cup X}(s,v) = d_{S \cup X}(s,v) = d(s,v)$. □

As Thorup noted [Tho99], Lemma 3.3 alone leads to a simple recursive procedure for computing SSSP; however, it makes no guarantee as to efficiency. The input to the procedure is an $(S, I)$-SAFE subgraph $X$; its only task is to compute the set $X^I$, which it performs with recursive calls (corresponding to Lemma 3.3(i) and (ii)) or directly if $X$ consists of a single vertex. There are essentially three major obstacles to making this general algorithm efficient: bounding the number of recursive calls, bounding the *time* to decide what those recursive calls are, and computing good $t$-partitions. Thorup gave a simple way to choose the $t$-partitions in integer-weighted graphs so that the number of recursive calls is $O(n)$. However, if adapted directly to the comparison-addition model, the time to decide which calls to make becomes $\Omega(n \log n)$; it amounts to the problem of implementing a general priority queue. We reduce the overhead for deciding which recursive calls to make to linear by using a "well balanced" hierarchy and a specialized priority queue for exploiting this kind of balance. Our techniques rely heavily on the graph being undirected and do not seem to generalize to directed graphs in any way.

As in other hierarchy-type algorithms, we generalize the distance and tentative distance notation from Dijkstra's algorithm to include not just single vertices but sets of vertices. If $X$ is a set of vertices (or associated with a set of vertices), then

$$(1) \qquad D(X) \overset{\mathbf{def}}{=} \min_{v \in X} D(v) \qquad \text{and} \qquad d(u,X) \overset{\mathbf{def}}{=} \min_{v \in X} d(u,v).$$

The procedure Generalized-Visit, given below, takes a vertex set $X$ that is $(S, I)$-safe and computes the distances to all vertices in $X^I$, placing these vertices in the set $S$ as their distances become known. We maintain Invariant 2.1 at all times. By Definition 3.1 we can compute the set $X^I$ without looking at parts of the graph outside of $S \cup X$. If $X = \{v\}$ happens to contain a single vertex, we can compute $X^I$ directly: if $D(v) \in I$, then $X^I = \{v\}$; otherwise it is $\emptyset$. For the general case, Lemma 3.3 says that we can compute $X^I$ by first finding a $t$-partition $\chi$ of $X$, then computing $X^I$ in phases. Let $I = I_1 \cup I_2 \cup \cdots \cup I_k$, where each subinterval is disjoint from the others and has width $t$, except perhaps $I_k$, which may be a leftover interval of width less than $t$. Let $S_i = S \cup X^{I_1} \cup \cdots \cup X^{I_i}$ and let $S_0 = S$. By the assumption that $X$ is $(S, I)$-safe and Lemma 3.3, each set in $\chi$ is $(S_i, I_{i+1})$-safe. Therefore, we can compute $S_1, S_2, \ldots, S_k = S \cup X^I$ with a series of recursive calls as follows. Assume that the current set of visited vertices is $S_i$. We determine $X^{I_{i+1}} = \bigcup_{Y \in \chi} Y^{I_{i+1}}$ with recursive calls of the form Generalized-Visit$(Y, I_{i+1})$, for every $Y \in \chi$ such that $Y^{I_{i+1}} \neq \emptyset$.

To start things off, we initialize the set $S$ to be empty, set the $D$-values (tentative distances) according to Invariant 2.1, and call Generalized-Visit$(V(G), [0, \infty))$. By the definition of safeness, $V(G)$ is clearly $(\emptyset, [0, \infty))$-safe. If Generalized-Visit works according to specification, when it completes $S = V(G)$ and Invariant 2.1 is satisfied, implying that $D(v) = d(s, v)$ for all vertices $v \in V(G)$.

```
Generalized-Visit($X$, $[a, b)$): A generalized hierarchy-type algorithm for real-weighted graphs.
        Input guarantee: $X$ is $(S, [a, b))$-safe and Invariant 2.1 is satisfied.

        Output guarantee: Invariant 2.1 is satisfied and $S_{post} = S_{pre} \cup X^{[a,b)}$,
        where $S_{pre}$ and $S_{post}$ are the set $S$ before and after the call.

    1. If $X$ contains one vertex, $X = \{v\}$, and $D(v) \in [a, b)$, then $D(v) =
       d_S(s, v) = d(s, v)$, where the first equality is by Invariant 2.1 and the
       second by the assumption that $X$ is $(S, [a, b))$-safe. Let $S := S \cup \{v\}$.
       Relax all edges incident on $v$, restoring Invariant 2.1, and return.
    2. Let $a' := a$
       While $a' < b$ and $X \not\subseteq S$
           Let $t > 0$ be any positive real
           Let $\chi = \{X_1, X_2, \ldots, X_k\}$ be an arbitrary $t$-partition of $X$
           Let $\chi' = \{X_i \in \chi \ :\ D(X_i) < \min\{a' + t, b\}$ and $X_i \not\subseteq S\}$
           For each $X_i \in \chi'$, Generalized-Visit($X_i$, $[a', \min\{a' + t, b\})$)
           $a' := \min\{a' + t, b\}$
```

Lemma 3.4. *If the input guarantees of* Generalized-Visit *are met, then after a call to* Generalized-Visit$(X, I)$*, Invariant* 2.1 *remains satisfied and* $X^I$ *is a subset of the visited vertices* $S$.

*Proof* (sketch). The base case, when $X$ is a single vertex, is simple to handle. Turning to the general case, we prove that each time the while statement is examined in step 2, $X$ is $(S, [a', b))$-safe for the *current* value of $S$ and $a'$; in what follows we will treat $S$ as a variable, not a specific vertex set. The first time through the while-loop in step 2, it follows from the input guarantee to Generalized-Visit that $X$ is $(S, [a', b))$-safe. Similarly, the input guarantee for all recursive calls holds by Lemma 3.3. However, to show that $X$ is $(S, [a', b))$-safe at the assignment $a' := \min\{a' + t, b\}$, by Definition 3.1 we must show $X^{[0, \min\{a'+t, b\})} \subseteq S$. We assume inductively that the output guarantee of any recursive call to Generalized-Visit is fulfilled; that is, upon the completion of Generalized-Visit$(X_i, [a', \min\{a' + t, b\}])$, $S$ includes the set $X_i^{[a', \min\{a'+t,b\})}$. Each time through the while-loop in step 2 Generalized-Visit makes recursive calls to all $Y \in \chi'$. To complete the proof we must show that for $Y \in \chi \backslash \chi'$, $Y^{[a', \min\{a'+t,b\})} \backslash S = \emptyset$. If $Y \in \chi \backslash \chi'$, it was because $D(Y) \geq \min\{a' + t, b\}$ or because $Y \subseteq S$, both of which clearly imply $Y^{[a', \min\{a'+t,b\})} \backslash S = \emptyset$. The output guarantee for Generalized-Visit is clearly satisfied if step 1 is executed; if step 2 is executed, then when the while-loop finishes, $X$ is either $(S, [b, b))$-safe or $X \subseteq S$, both implying $X^{[0,b)} \in S$. □

Generalized-Visit can be simplified in a few minor ways. It can be seen that in step 1 we do not need to check whether $D(v) \in [a, b)$; the recursive call would not have taken place were this not the case. In step 2 the final line can be shortened to $a' := a' + t$. However, we cannot change all occurrences of $\min\{a' + t, b\}$ to $a' + t$ because this is crucial to the procedure's correctness. It is not assumed (nor can it be guaranteed) that $t$ divides $(b - a)$, so the procedure must be prepared to deal with fractional intervals of width less than $t$. In section 4 we show that for a *proper* hierarchy this fractional interval problem does not arise.

## 4. Efficient implementations of Generalized-Visit.

We propose two implementations of the Generalized-Visit algorithm, called Visit and Visit-B. The time bound claimed in Theorem 1.1 is proved by analyzing Visit, given later in this section. Although Visit is asymptotically fast, it seems too impractical for a real-world implementation. In section 4.5 we give the Visit-B implementation of Generalized-Visit, which uses fewer specialized data structures. The asymptotic running time of Visit-B is just a little slower than that of Visit.

Visit and Visit-B differ from Generalized-Visit in their input/output specification only slightly. Rather than accepting a set of vertices, as Generalized-Visit does, our implementations (like [Tho99, Hag00, Pet04, Pet02b]) accept a *hierarchy node* $x$, which represents a set of vertices. Both of our implementations work correctly for any *proper* hierarchy $\mathcal{H}$, defined below. We prove bounds on their running times as a function of $m, n$, and a certain function of $\mathcal{H}$ (which is different for Visit and Visit-B). In order to compute SSSP in near-linear time the proper hierarchy $\mathcal{H}$ must satisfy certain *balance* conditions, which are the same for Visit and Visit-B. In section 5 we give the requisite properties of a balanced hierarchy and show how to construct a balanced proper hierarchy in $O(\textsc{mst}(m, n) + \min\{n \log n,\ n \log \log r\})$ time. Definition 4.1, given next, describes exactly what is meant by hierarchy and proper hierarchy.

Definition 4.1. *A* hierarchy *is a rooted tree whose leaf nodes correspond to graph vertices. If $x$ is a hierarchy node, then $p(x)$ is its parent, $\textsc{deg}(x)$ is the number of children of $x$, $V(x)$ is the set of descendant leaves (or the equivalent graph vertices), and $\textsc{diam}(x)$ is an upper bound on the diameter of $V(x)$ (where the diameter of $V(x)$ is defined to be $\max_{u,v \in V(x)} d(u, v)$). Each node $x$ is given a value $\textsc{norm}(x)$. A hierarchy is* proper *if the following hold:*

(i) $\textsc{norm}(x) \leq \textsc{norm}(p(x))$,

(ii) *either* $\textsc{norm}(p(x))/\textsc{norm}(x)$ *is an integer or* $\textsc{diam}(x) < \textsc{norm}(p(x))$,

(iii) $\textsc{deg}(x) \neq 1$,

(iv) *if $x_1, \ldots, x_{\textsc{deg}(x)}$ are the children of $x$, then $\{V(x_i)\}_i$ is a $\textsc{norm}(x)$-partition of $V(x)$. (Refer to Definition 3.2 for the meaning of "$\textsc{norm}(x)$-partition.")*

Part (iv) of Definition 4.1 is the crucial one for computing shortest paths. Part (iii) guarantees that a proper hierarchy has $O(n)$ nodes. The second part of (ii) is admittedly a little strange. It allows us to replace all occurrences of $\min\{a + t, b\}$

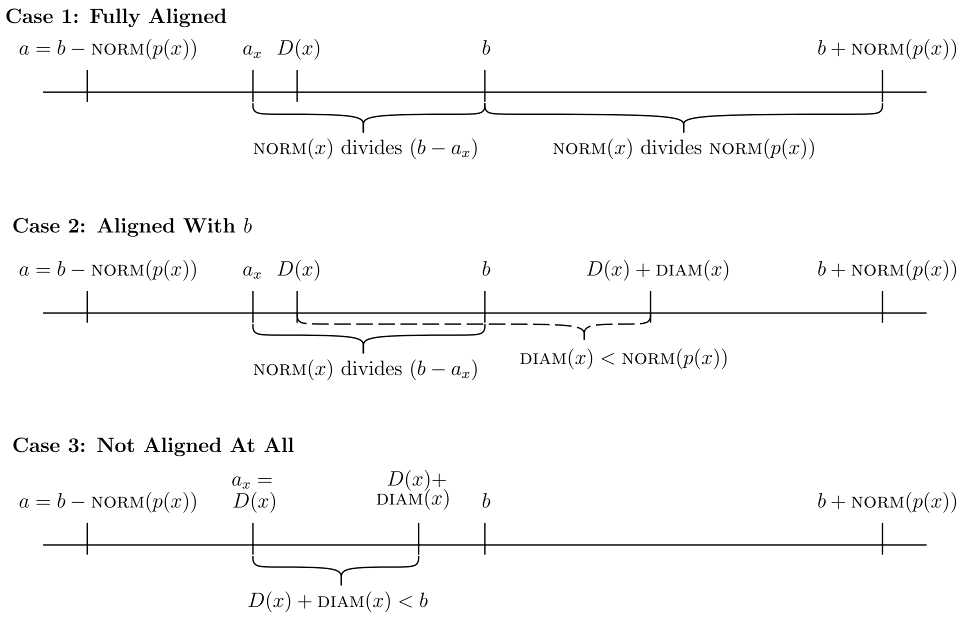
in GENERALIZED-VISIT with just $a + t$, which greatly simplifies the analysis of our algorithms. Part (i) will be useful when bounding the total number of recursive calls to our algorithms.

**4.1.** VISIT**.** Consider the VISIT procedure given below. We prove that VISIT correctly computes SSSPs by demonstrating that it is an implementation of GENERALIZED-VISIT, which was already proved correct.

```
VISIT($x$, $[a, b)$).

    Input: $x$ is a node in a proper hierarchy $\mathcal{H}$; $V(x)$ is $(S, [a, b))$-SAFE and
    Invariant 2.1 is satisfied.

    Output guarantee: Invariant 2.1 is satisfied and $S_{post} = S_{pre} \cup V(x)^{[a,b)}$,
    where $S_{pre}$ and $S_{post}$ are the set $S$ before and after the call.

 1. If $x$ is a leaf and $D(x) \in [a, b)$, then let $S := S \cup \{x\}$, relax all edges
    incident on $x$, restoring Invariant 2.1, and return.
 2. If VISIT$(x, \cdot)$ is being called for the first time, create a bucket array of
    $\lceil \text{DIAM}(x)/\text{NORM}(x) \rceil + 1$ buckets. Bucket $i$ represents the interval

        $[a_x + i \cdot \text{NORM}(x),\ a_x + (i + 1) \cdot \text{NORM}(x)),$

    where  $a_x = \begin{cases} D(x) & \text{if } D(x) + \text{DIAM}(x) < b, \\ b - \lceil \frac{b - D(x)}{\text{NORM}(x)} \rceil \text{NORM}(x) & \text{otherwise.} \end{cases}$

    We initialize $a' := a_x$ and insert all the children of $x$ in $\mathcal{H}$ into the bucket
    array.

    The bucket invariant: A node $y \in \mathcal{H}$ in $x$'s bucket array appears (log-
    ically) in the bucket whose interval spans $D(y)$. If $\{x_i\}$ are the set of
    bucketed nodes, then $\{V(x_i)\}$ is a NORM$(x)$-partition of $V(x)$.

 3. While $a' < b$ and $V(x) \not\subseteq S$
        While $\exists y$ in bucket $[a',\ a' + \text{NORM}(x))$ s.t. $\text{NORM}(y) = \text{NORM}(x)$
            Remove $y$ from the bucket array
            Insert $y$'s children in $\mathcal{H}$ in the bucket array
        For each $y$ in bucket $[a', a' + \text{NORM}(x))$
        and each $y$ such that $D(y) < a'$ and $V(y) \not\subseteq S$
            VISIT$(y, [a', a' + \text{NORM}(x)))$
        $a' := a' + \text{NORM}(x)$
```

In step 2 of GENERALIZED-VISIT we let $\chi$ be any arbitrary $t$-partition of the subset of vertices given as input. In VISIT the input is a hierarchy node $x$, and the associated vertex set is $V(x)$. We represent the $t$-partition of $V(x)$ (where $t = \text{NORM}(x)$) by the set of bucketed $\mathcal{H}$-nodes $\{x_i\}_i$ (see step 2), where the sets $\{V(x_i)\}_i$ partition $V(x)$. Clearly the $\{x_i\}_i$ are descendants of $x$. The set $\{x_i\}_i$ will begin as $x$'s children, though later on $\{x_i\}_i$ may contain a mixture of children of $x$, grandchildren of $x$, and so on.

Consider the inner while-loop in step 3. Assuming inductively that the bucketed $\mathcal{H}$-nodes represent a NORM$(x)$-partition of $V(x)$, if $y$ is a bucketed node and $\text{NORM}(y) = \text{NORM}(x)$, then replacing $y$ by its children in the bucket array produces a new NORM$(x)$-partition. This follows from the definitions of $t$-partitions and proper



Case 1: Fully Aligned
$a = b - \text{NORM}(p(x))$
$a_x$
$D(x)$
$b$
$b + \text{NORM}(p(x))$
NORM($x$) divides $(b - a_x)$
NORM($x$) divides NORM($p(x)$)
Case 2: Aligned With $b$
$a = b - \text{NORM}(p(x))$
$a_x$
$D(x)$
$b$
$D(x) + \text{DIAM}(x)$
$b + \text{NORM}(p(x))$
NORM($x$) divides $(b - a_x)$
$\text{DIAM}(x) < \text{NORM}(p(x))$
Case 3: Not Aligned At All
$a = b - \text{NORM}(p(x))$
$a_x = D(x)$
$D(x)+\text{DIAM}(x)$
$b$
$b + \text{NORM}(p(x))$
$D(x) + \text{DIAM}(x) < b$

FIG. 1. *First observe that when $a_x$ is initialized we have $D(x) \geq a_x \geq a$, as in the figure. If $a_x$ is chosen such that* NORM($x$) *divides $(b - a_x)$, then by Definition* 4.1(ii) *either* NORM($x$) *divides* NORM($p(x)$) *(which puts us in Case* 1*) or* $\text{DIAM}(x) < \text{NORM}(p(x))$ *(putting us in Case* 2*); that is,* NORM($x$) *does not divide $(b + \text{NORM}(p(x)) - a_x)$, but it does not matter since we'll never reach $b + \text{NORM}(p(x))$ anyway. If $a_x$ is chosen so that* NORM($x$) *does not divide $(b - a_x)$, then $a_x = D(x)$ and $D(x) + \text{DIAM}(x) < b$ (putting us in Case* 3*), meaning we will never reach $b$. Note that by the definition of* DIAM($x$) *(Definition* 4.1*) and Invariant* 2.1*, for any vertex in $u \in V(x)$ we have $d(s, u) \leq d(s, x) + \text{DIAM}(x) \leq D(x) + \text{DIAM}(x)$.*

hierarchies (Definitions 3.2 and 4.1). Since the bucketed nodes form a NORM($x$)-partition, one can easily see that the recursive calls in step 3 of VISIT correspond to the recursive calls in GENERALIZED-VISIT. However, their interval arguments are different. We sketch below why this change does not affect correctness.

In GENERALIZED-VISIT the intervals passed to recursive calls are of the form $[a', \min\{a' + t, b\})$, whereas in VISIT they are $[a', a' + t) = [a', a' + \text{NORM}(x))$. We will argue why $a' + t = a' + \text{NORM}(x)$ is never more than $b$. The main idea is to show that we are always in one of the three cases portrayed in Figure 1.

If NORM($x$) divides NORM($p(x)$) and $a_x$ is chosen in step 2 so that $t = \text{NORM}(x)$ divides $(b - a_x)$, then we can freely substitute the interval $[a', a' + t)$ for $[a', \min\{a' + t, b\})$ since they will be identical. Note that in our algorithm $(b - a) = \text{NORM}(p(x))$.[6] The problems arise when NORM($x$) does not divide either NORM($p(x)$) or $(b - a_x)$. In order to prove the correctness of VISIT we must show that the input guarantee (regarding SAFE-ness) is satisfied for each recursive call. We consider two cases: when we are in the first recursive call to VISIT($x, \cdot$) and any subsequent call. Suppose we are in the first recursive call to VISIT($x, \cdot$). By our choice of $a_x$ in step 2, either $b = a_x + q \cdot \text{NORM}(x)$ for some integer $q$, or $b > D(x) + \text{DIAM}(x) = a_x + \text{DIAM}(x)$. If it is the first case, each time the outer while-loop is entered we have $a' < b$, which,

[6]Strictly speaking, this does not hold for the initial call because in this case, $x = \text{ROOT}(\mathcal{H})$ is the root of the hierarchy $\mathcal{H}$ and there is no such node $p(x)$. The argument goes through just fine if we let $p(\text{ROOT}(\mathcal{H}))$ denote a dummy node such that $\text{NORM}(p(\text{ROOT}(\mathcal{H}))) = \infty$.

since $q$ is integral, implies $\min\{a' + \textsc{norm}(x), b\} = a' + \textsc{norm}(x)$. Now consider the second case, where $b > D(x) + \textsc{diam}(x) = a_x + \textsc{diam}(x)$, and one of the recursive calls $\textsc{Visit}(y, [a', a' + \textsc{norm}(x)))$ made in step 3. By Lemma 3.3, $V(y)$ is $(S, [a', \min\{a' + \textsc{norm}(x), b\}))$-SAFE, and it is actually $(S, [a', a' + \textsc{norm}(x)))$-SAFE as well because $b > D(x) + \textsc{diam}(x)$, implying $V(y)^{[b,\infty)} \subseteq V(x)^{[b,\infty)} = \emptyset$. (Recall from Definition 4.1 that for any $u \in V(x)$, $\textsc{diam}(x)$ satisfies $d(s, x) \leq d(s, u) \leq d(s, x) + \textsc{diam}(x) \leq D(x) + \textsc{diam}(x)$.) Now consider a recursive call $\textsc{Visit}(x, [a, b))$ that is *not* the first call to $\textsc{Visit}(x, \cdot)$. Then by Definition 4.1(ii), either $(b - a) = \textsc{norm}(p(x))$ is a multiple of $\textsc{norm}(x)$ or $a + \textsc{diam}(x) < b$; these are identical to the two cases treated above.

There are two data structural problems that need to be solved in order to efficiently implement VISIT. First, we need a way to compute the tentative distances of hierarchy nodes, i.e., the $D$-values as defined in (1) in section 3. For this problem we use an improved version of Gabow's split-findmin structure [G85a]. The other problem is efficiently implementing the various bucket arrays, which we solve with a new structure called the *bucket-heap.* The specifications for these two structures are discussed below, in sections 4.2 and 4.3, respectively. The interested reader can refer to Appendices A and B for details about our implementations of split-findmin and the bucket-heap, and for proofs of their respective complexities.

**4.2. The split-findmin structure.** The split-findmin structure operates on a collection of disjoint sequences, consisting in total of $n$ elements, each with an associated key. The idea is to maintain the smallest key in each sequence under the following operations.

| | |
|---|---|
| split($x$): | Split the sequence containing $x$ into two sequences: the elements up to and including $x$, and the rest. |
| decrease-key($x, \kappa$): | Set key($x$) $= \min\{\text{key}(x), \kappa\}$. |
| findmin($x$): | Return the element with minimum key in $x$'s sequence. |

Theorem 4.2, given below, establishes some new bounds on the problem that are just slightly better than Gabow's original data structure [G85a]. Refer to Appendix B for a proof. Thorup [Tho99] gave a similar data structure for integer keys in the RAM model that runs in linear time. It relies on the RAM's ability to sort small sets of integers in linear time [FW93].

THEOREM 4.2. *The split-findmin problem can be solved on a pointer machine in $O(n+m\alpha)$ time while making only $O(n+m\log\alpha)$ comparisons, where $\alpha = \alpha(m, n)$ is the inverse-Ackermann function. Alternatively, split-findmin can be solved on a RAM in time $\Theta(\textsc{split-findmin}(m, n))$, where $\textsc{split-findmin}(m, n) = O(n+m\log\alpha)$ is the decision-tree complexity of the problem.*

We use the split-findmin structure to maintain $D$-values as follows. In the beginning there is one sequence consisting of the $n$ leaves of $\mathcal{H}$ in an order consistent with some depth-first search traversal of $\mathcal{H}$. For any leaf $v$ in $\mathcal{H}$ we maintain, by appropriate decrease-key operations, that key($v$) $= D(v)$. During execution of VISIT we will say an $\mathcal{H}$-node is *unresolved* if it lies in another node's bucket array but its tentative distance ($D$-value) is not yet finalized. The $D$-value of an $\mathcal{H}$-node becomes finalized, in the sense that it never decreases again, during step 3 of VISIT, either by being removed from some bucket array or passed, for the first time, to a recursive call of VISIT. (It follows from Definition 3.1 and Invariant 2.1 that $D(y) = d(s, y)$ at the first recursive call to $y$.) One can verify a couple properties of the unresolved nodes.

First, each unvisited leaf has exactly one unresolved ancestor. Second, to implement Visit we need only query the $D$-values of unresolved nodes. Therefore, we maintain that for each unresolved node $y$, there is some sequence in the split-findmin structure corresponding to $V(y)$, the descendants of $y$. Now suppose that a previously unresolved node $y$ is *resolved* in step 3 of Visit. The $\mathrm{DEG}(y)$ children of $y$ will immediately become unresolved, so to maintain our correspondence between sequences and unresolved nodes, we perform $\mathrm{DEG}(y) - 1$ split operations on $y$'s sequence, so that the resulting subsequences correspond to $y$'s children.

We remark that the split-findmin structure we use can be simplified slightly because we know in advance where the splits will occur. However, this knowledge does not seem to affect the asymptotic complexity of the problem.

## 4.3. The bucket-heap.

We now turn to the problem of efficiently implementing the bucket array used in Visit. Because of the information-theoretic bottleneck built into the comparison-addition model, we cannot always bucket nodes in constant time: each comparison extracts at most one bit of information, whereas properly bucketing a node in $x$'s bucket array requires us to extract up to $\log(\mathrm{DIAM}(x)/\mathrm{NORM}(x))$ bits of information. Thorup [Tho99] and Hagerup [Hag00] assume integer edge lengths and the RAM model and therefore do not face this limitation. We now give the specification for the *bucket-heap*, a structure that supports the bucketing operations of Visit. This structure logically operates on a sequence of buckets; however, our implementation is really a simulation of the logical structure. Lemma 4.3, proved in Appendix A, bounds the complexity of our implementation of the bucket-heap.

| | |
|---|---|
| create$(\mu, \delta)$: | Create a new bucket-heap whose buckets are associated with intervals $[\delta, \delta + \mu), [\delta + \mu, \delta + 2\mu), [\delta + 2\mu, \delta + 3\mu), \ldots$. An item $x$ lies in the bucket whose interval spans key$(x)$. All buckets are initially *open*. |
| insert$(x, \kappa)$: | Insert a new item $x$ with key$(x) = \kappa$. |
| decrease-key$(x, \kappa)$: | Set key$(x) = \min\{\text{key}(x), \kappa\}$. It is guaranteed that $x$ is not moved to a closed bucket. |
| enumerate: | Close the first open bucket and enumerate its contents. |

Lemma 4.3. *Let $\Delta_x \geq 1$ denote the number of buckets between the first open bucket at the time of $x$'s insertion and the bucket from which $x$ was enumerated. The bucket-heap can be implemented on a pointer machine to run in $O(N + \sum_x \log \Delta_x)$ time, where $N$ is the number of operations.*

When $\mathrm{Visit}(x, \cdot)$ is called for the first time, we initialize the bucket-heap at $x$ with a call to create$(\mathrm{NORM}(x),\ a_x)$, followed by a number of insert operations for each of $x$'s children, where the key of a child is its $D$-value. Here $a_x$ is the beginning of the real interval represented by the bucket array, and $\mathrm{NORM}(x)$ the width of each bucket. Every time the $D$-value of a bucketed node decreases, which can easily be detected with the split-findmin structure, we perform a decrease-key on the corresponding item in the bucket-heap. We usually refer to buckets not by their cardinal number but by their associated real interval, e.g., bucket $[a_x, a_x + \mathrm{NORM}(x))$.

## 4.4. Analysis of Visit.

In this section we bound the time required to compute SSSP with Visit as a function of $m$, $n$, and the given hierarchy $\mathcal{H}$. We will see later that the dominant term in this running time corresponds to the split-findmin structure, whose complexity is no more than $O(m \log \alpha)$ but could turn out to be linear.

LEMMA 4.4. *Let $\mathcal{H}$ be a proper hierarchy. Computing SSSPs with* VISIT *on $\mathcal{H}$ takes time $O(\text{SPLIT-FINDMIN}(m,n) + \phi(\mathcal{H}))$, where $\text{SPLIT-FINDMIN}(m,n)$ is the complexity of the split-findmin problem and*

$$\phi(\mathcal{H}) = \sum_{\substack{x\in\mathcal{H} \text{ such that} \\ \text{NORM}(x)\neq\text{NORM}(p(x))}} \frac{\text{DIAM}(x)}{\text{NORM}(x)} + \sum_{x\in\mathcal{H}} \log\left(\frac{\text{DIAM}(p(x))}{\text{NORM}(p(x))} + 1\right).$$

*Proof.* The $\text{SPLIT-FINDMIN}(m,n)$ term represents the time to relax edges (in step 1) and update the relevant $D$-values of $\mathcal{H}$-nodes, as described in section 4.2. Except for the costs associated with updating $D$-values, the overall time of VISIT is linear in the number of recursive calls and the bucketing costs. The two terms of $\phi(\mathcal{H})$ represent these costs. Consider the number of calls to $\text{VISIT}(x, I)$ for a particular $\mathcal{H}$-node $x$. According to step 3 of VISIT, there will be *zero* calls to $x$ unless $\text{NORM}(x) \neq \text{NORM}(p(x))$. If it is the case that $\text{NORM}(x) \neq \text{NORM}(p(x))$, then for all recursive calls on $x$, the given interval $I$ will have the same width: $\text{NORM}(z)$ for some ancestor $z$ of $x$. By Definition 4.1(i), $\text{NORM}(z) \geq \text{NORM}(x)$, and therefore the number of such recursive calls on $x$ is $\leq \text{DIAM}(x)/\text{NORM}(x) + 2$; the extra 2 counts the first and last recursive calls, which may cover negligible parts of the interval $[d(s,x), d(s,x) + \text{DIAM}(x)]$. By Definition 4.1(iii), $|\mathcal{H}| < 2n$, and therefore the total number of recursive calls is bounded by $4n + \sum_x \text{DIAM}(x)/\text{NORM}(x)$, where the summation is over $\mathcal{H}$ nodes whose NORM-values differ from their parents' NORM-values.

Now consider the bucketing costs of VISIT if implemented with the bucket-heap. According to steps 2 and 3, a node $y$ is bucketed either because $\text{VISIT}(p(y), \cdot)$ was called for the first time, or its parent $p(y)$ was removed from the first open bucket (of some bucket array), say bucket $[a, a + \text{NORM}(p(y)))$. In either case, this means that $d(s, p(y)) \in [a, a + \text{NORM}(p(y)))$ and that $d(s, y) \in [a, a + \text{NORM}(p(y)) + \text{DIAM}(p(y)))$. To use the terminology of Lemma 4.3, $\Delta_y \leq \lceil \text{DIAM}(p(y))/\text{NORM}(p(y)) \rceil$, and the total bucketing costs would be #(buckets scanned) + #(insertions) + #(dec-keys) + $\sum_x \log(\text{DIAM}(p(x))/\text{NORM}(p(x)) + 1)$, which is $O(\phi(\mathcal{H}) + m + n)$. ☐

In section 5 we give a method for constructing a proper hierarchy $\mathcal{H}$ such that $\phi(\mathcal{H}) = O(n)$. This bound together with Lemma 4.4 shows that we can compute SSSP in $O(\text{SPLIT-FINDMIN}(m,n))$ time, given a suitable hierarchy. Asymptotically speaking, this bound is the best we are able to achieve. However, the promising experimental results of a simplified version of our algorithm [PRS02] have led us to design an alternate implementation of GENERALIZED-VISIT that is both theoretically fast and easier to code.

### 4.5. A practical implementation of GENERALIZED-VISIT.

In this section we present another implementation of GENERALIZED-VISIT, called VISIT-B. Although VISIT-B is a bit slower than VISIT in the asymptotic sense, it has other advantages. Unlike VISIT, VISIT-B treats all internal hierarchy nodes in the same way and is generally more streamlined. VISIT-B also works with any optimal off-the-shelf priority queue, such as a Fibonacci heap [FT87]. We will prove later that the asymptotic running time of VISIT-B is $O(m + n\log^* n)$. Therefore, if $m/n = \Omega(\log^* n)$, both VISIT and VISIT-B run in optimal $O(m)$ time.

The pseudocode for VISIT-B is given as follows.

```
Visit-B($x$, $[a,b)$).

        Input: $x$ is a node in a proper hierarchy $\mathcal{H}$; $V(x)$ is $(S,[a,b))$-safe and
        Invariant 2.1 is satisfied.

        Output guarantee: Invariant 2.1 is satisfied and $S_{post} = S_{pre} \cup V(x)^{[a,b)}$,
        where $S_{pre}$ and $S_{post}$ are the set $S$ before and after the call.

    1. If $x$ is a leaf and $D(x) \in [a,b)$, then let $S := S \cup \{x\}$, relax all edges
       incident on $x$, restoring Invariant 2.1, and return.
    2. If Visit-B$(x,\cdot)$ is being called for the first time, put $x$'s children in $\mathcal{H}$
       in a heap associated with $x$, where the key of a node is its $D$-value.
       Choose $a_x$ as in Visit and initialize $a' := a_x$ and $\chi := \emptyset$.
    3. While $a' < b$ and either $\chi$ or $x$'s heap is nonempty,
           While there exists a $y$ in $x$'s heap with $D(y) \in [a', a' + \text{norm}(x))$
               Remove $y$ from the heap
               $\chi := \chi \cup \{y\}$
           For each $y \in \chi$
               Visit-B$(y, [a', a' + \text{norm}(x)))$
               If $V(y) \subseteq S$, then set $\chi := \chi \backslash \{y\}$
           $a' := a' + \text{norm}(x)$
```

The proof of correctness for Visit-B follows the same lines as that for Visit. It is easy to establish that before the for-loop in step 3 is executed, $\chi = \{y : p(y) = x, D(y) < a' + \text{norm}(x), \text{ and } V(y) \not\subseteq S\}$, so Visit-B is actually a more straightforward implementation of Generalized-Visit than Visit. In Visit-B the norm$(x)$-partition for $x$ corresponds to $x$'s children, whereas in Visit the partition begins with $x$'s children but is decomposed progressively.

Lemma 4.5. *Let $\mathcal{H}$ be a proper hierarchy. Computing SSSPs with* Visit-B *on $\mathcal{H}$ takes time $O(\text{split-findmin}(m,n) + \psi(\mathcal{H}))$, where split-findmin$(m,n)$ is the complexity of the split-findmin problem and*

$$\psi(\mathcal{H}) \;=\; \sum_{x \in \mathcal{H}} \left( \frac{\text{diam}(x)}{\text{norm}(x)} + \text{deg}(x) \log \text{deg}(x) \right).$$

*Proof.* The split-findmin term plays the same role in Visit-B as in Visit. Visit-B is different than Visit in that it makes recursive calls on *all* hierarchy nodes, not just those with different norm-values than their parents. Using the same argument as in Lemma 4.5, we can bound the number of recursive calls of the form Visit-B$(x,\cdot)$ as diam$(x)$/norm$(x) + 2$; this gives the first summation in $\psi(\mathcal{H})$. Assuming an optimal heap is used (for example, a Fibonacci heap [FT87]), all decrease-keys take $O(m)$ time, and all deletions take $\sum_x \text{deg}(x) \log \text{deg}(x)$ time. The bound on deletions follows since each of the deg$(x)$ children of $x$ is inserted into and deleted from $x$'s heap at most once. ∎

In section 5 we construct a hierarchy $\mathcal{H}$ such that $\psi(\mathcal{H}) = \Theta(n\log^* n)$, implying an overall bound on Visit-B of $O(m + n\log^* n)$, since split-findmin$(m,n) = O(m\alpha(m,n)) = O(m + n\log^* n)$. Even though $\psi(\mathcal{H}) = \Omega(n\log^* n)$ in the worst case, we are only able to construct very contrived graphs for which this lower bound is tight.

**5. Efficient construction of balanced hierarchies.** In this section we construct a hierarchy that works well for both Visit and Visit-B. The construction procedure has three distinct phases. In phase 1 we find the graph's minimum spanning tree, denoted $M$, and classify its edges by length. This classification immediately induces a coarse hierarchy, denoted $\mathcal{H}_0$, which is analogous to the *component hierarchy* defined by Thorup [Tho99] for integer-weighted graphs. Although $\mathcal{H}_0$ is proper, using it to run Visit or Visit-B may result in a slow SSSP algorithm. In particular, $\phi(\mathcal{H}_0)$ and $\psi(\mathcal{H}_0)$ can easily be $\Theta(n \log n)$, giving no improvement over Dijkstra's algorithm. Phase 2 facilitates phase 3, in which we produce a *refinement* of $\mathcal{H}_0$, called $\mathcal{H}$; this is the "well balanced" hierarchy we referred to earlier. The refined hierarchy $\mathcal{H}$ is constructed so as to minimize the $\phi(\mathcal{H})$ and $\psi(\mathcal{H})$ terms in the running times of Visit and Visit-B. In particular, $\phi(\mathcal{H})$ will be $O(n)$, and $\psi(\mathcal{H})$ will be $O(n\log^* n)$. Although $\mathcal{H}$ could be constructed directly from $M$ (the graph's minimum spanning tree), we would not be able to prove the time bound of Theorem 1.1 using this method. The purpose of phase 2 is to generate a collection of small auxiliary graphs that—loosely speaking—capture the structure and edge lengths of certain subtrees of the minimum spanning tree. Using the auxiliary graphs in lieu of $M$, we are able to construct $\mathcal{H}$ in phase 3 in $O(n)$ time.

In section 5.1 we define all the notation and properties used in phases 1, 2, and 3 (sections 5.2, 5.3, and 5.4, respectively). In section 5.5 we prove that $\phi(\mathcal{H}) = O(n)$ and $\psi(\mathcal{H}) = O(n\log^* n)$.

## 5.1. Some definitions and properties.

**5.1.1. The coarse hierarchy.** Our refined hierarchy $\mathcal{H}$ is derived from a *coarse hierarchy* $\mathcal{H}_0$, which is defined here and in section 5.2. Although $\mathcal{H}_0$ is typically very simple to describe, the general definition of $\mathcal{H}_0$ is rather complicated since it must take into account certain extreme circumstances. $\mathcal{H}_0$ is defined w.r.t. an increasing sequence of norm-values: $\text{NORM}_1, \text{NORM}_2, \ldots$, where all edge lengths are at least as large as $\text{NORM}_1$. (Typically $\text{NORM}_{i+1} = 2 \cdot \text{NORM}_i$; however, this is not true in general.) We will say that an edge $e$ is at level $i$ if $\ell(e) \in [\text{NORM}_i, \text{NORM}_{i+1})$, or alternatively, we may write $\text{NORM}(e) = \text{NORM}_i$ to express that $e$ is at level $i$. A level $i$ subgraph is a maximal connected subgraph restricted to edges with level $i$ or less, that is, with length strictly less than $\text{NORM}_{i+1}$. Therefore, the level zero subgraphs consist of single vertices. A level $i$ node in $\mathcal{H}_0$ corresponds to a nonredundant level $i$ subgraph, where a level $i$ subgraph is redundant if it is also a level $i-1$ subgraph. This nonredundancy property guarantees that all nonleaf $\mathcal{H}_0$-nodes have at least two children. The ancestor relationship in $\mathcal{H}_0$ should be clear: $x$ is an ancestor of $y$ if and only if the subgraph of $y$ is contained in the subgraph of $x$, i.e., $V(y) \subseteq V(x)$. The leaves of $\mathcal{H}_0$ naturally correspond to graph vertices, and the internal nodes to subgraphs. The coarse hierarchy $\mathcal{H}_0$ clearly satisfies Definition 4.1(i), (iii), (iv); however, we have to be careful in choosing the norm-values if we want it to be a *proper* hierarchy, that is, for it to satisfy Definition 4.1(ii) as well. Our method for choosing the norm-values is deferred to section 5.2.

**5.1.2. The minimum spanning tree.** By the *cut property* of minimum spanning trees (see [CLRS01, PR02c]) the $\mathcal{H}_0$ w.r.t. $G$ is identical to the $\mathcal{H}_0$ w.r.t. $M$, the minimum spanning tree (MST) of $G$. Therefore, the remainder of this section is mainly concerned with $M$, not the graph itself. If $X \subseteq V(G)$ is a set of vertices, we let $M(X)$ be the minimal connected subtree of $M$ containing $X$. Notice that $M(X)$ can include vertices outside of $X$. Later on we will need $M$ to be a rooted tree in order to talk coherently about a vertex's parent, ancestors, children, and so on. Assume that $M$ is rooted at an arbitrary vertex. The notation $\text{ROOT}(M(X))$ refers to the root of the subtree $M(X)$.

**5.1.3. Mass and diameter.** The MASS of a vertex set $X \subseteq V(G)$ is defined as

$$\textsc{mass}(X) \overset{\mathbf{def}}{=} \sum_{e \in E(M(X))} \ell(e).$$

Extending this notation, we let $M(x) = M(V(x))$ and $\textsc{mass}(x) = \textsc{mass}(V(x))$, where $x$ is a node in *any* hierarchy. Since the MST path between two vertices in $M(x)$ is an upper bound on the shortest path between them, $\textsc{mass}(x)$ is an upper bound on the diameter of $V(x)$. Recall from Definition 4.1 that $\textsc{diam}(x)$ denoted any upper bound on the diameter of $V(x)$; henceforth, we will freely substitute $\textsc{mass}(x)$ for $\textsc{diam}(x)$.

**5.1.4. Refinement of the coarse hierarchy.** We will say that $\mathcal{H}$ is a *refinement* of $\mathcal{H}_0$ if all nodes in $\mathcal{H}_0$ are also represented in $\mathcal{H}$. An equivalent definition, which provides us with better imagery, is that $\mathcal{H}$ is derived from $\mathcal{H}_0$ by *replacing* each node $x \in \mathcal{H}_0$ with a rooted subhierarchy $H(x)$, where the root of $H(x)$ corresponds to (and is also referred to as) $x$ and the leaves of $H(x)$ correspond to the children of $x$ in $\mathcal{H}_0$. Consider a refinement $\mathcal{H}$ of $\mathcal{H}_0$ where each internal node $y$ in $H(x)$ satisfies $\textsc{deg}(y) \neq 1$ and $\textsc{norm}(y) = \textsc{norm}(x)$. One can easily verify from Definitions 3.2 and 4.1 that if $\mathcal{H}_0$ is a proper hierarchy, so too is $\mathcal{H}$. Of course, in order for $\phi(\mathcal{H})$ and $\psi(\mathcal{H})$ to be linear or near-linear, $H(x)$ must satisfy certain properties. In particular, it must be sufficiently short and balanced. By balanced we mean that a node's mass should not be *too* much smaller than its parent's mass.

**5.1.5. Lambda values.** We will use the following $\lambda$-values in order to quantify precisely our notion of balance:

$$\lambda_0 = 0, \quad \lambda_1 = 12 \quad \text{and} \quad \lambda_{q+1} = 2^{\lambda_q \cdot 2^{-q}}.$$

Lemma 5.1 gives a lower bound on the growth of the $\lambda$-values; we give a short proof before moving on.

LEMMA 5.1. $\min\{q : \lambda_q \geq n\} \leq 2\log^* n$.

*Proof.* Let $\mathcal{S}_q$ be a stack of $q$ twos; for example, $\mathcal{S}_3 = 2^{2^2} = 16$. We will prove that $\lambda_q \geq \mathcal{S}_{\lfloor q/2 \rfloor}$, giving the lemma. One can verify that this statement holds for $q \leq 9$. Assume that it holds for all $q' \leq q$.

$$\begin{aligned}
\lambda_{q+1} &= 2^{2^{\lambda_{q-1} \cdot 2^{-(q-1)}} 2^{-q}} && \{\text{definition of } \lambda_{q+1}\} \\
&\geq 2^{2^{\mathcal{S}_{\lfloor (q-1)/2 \rfloor} \cdot 2^{-(q-1)} - q}} && \{\text{inductive assumption}\} \\
&\geq 2^{2^{\mathcal{S}_{\lfloor (q-1)/2 \rfloor - 1}}} = \mathcal{S}_{\lfloor (q+1)/2 \rfloor} && \{\text{holds for } q \geq 9\}.
\end{aligned}$$

The third line follows from the inequality $\mathcal{S}_{\lfloor (q-1)/2 \rfloor} \cdot 2^{-(q-1)} - q \geq \mathcal{S}_{\lfloor (q-1)/2 \rfloor - 1}$, which holds for $q \geq 9$. $\square$

**5.1.6. Ranks.** Recall from section 5.1.4 that our refined hierarchy $\mathcal{H}$ is derived from $\mathcal{H}_0$ by replacing each node $x \in \mathcal{H}_0$ with a subhierarchy $H(x)$. We assign to all nodes in $H(x)$ a nonnegative integer *rank*. The analysis of our construction would become very simple if for every rank $j$ node $y$ in $H(x)$, $\textsc{mass}(y) = \lambda_j \cdot \textsc{norm}(x)$. Although this is our ideal situation, the nature of our construction does not allow us to place any nontrivial lower or upper bounds on the mass of $y$. We will assign ranks in order to satisfy Property 5.1, given below, which ensures us a sufficiently good approximation to the ideal. It is mainly the internal nodes of $H(x)$ that can have subideal ranks; we assign ranks to the leaves of $H(x)$ (representing children of $x$ in $\mathcal{H}_0$) to be as close to the ideal as possible.

We should point out that the assignment of ranks is mostly for the purpose of analysis. Rank information is never stored explicitly in the hierarchy nodes, nor is rank information used, implicitly or explicitly, in the computation of shortest paths. We only refer to ranks in the construction of $\mathcal{H}$ and when analyzing their effect on the $\phi$ and $\psi$ functions.

PROPERTY 5.1. *Let $x \in \mathcal{H}_0$ and $y, z \in H(x) \subseteq \mathcal{H}$.*

(a) *If $y$ is an internal node of $H(x)$, then* $\textsc{norm}(y) = \textsc{norm}(x)$ *and* $\textsc{deg}(y) > 1$.

(b) *If $y$ is a leaf of $H(x)$ (i.e., a child of $x$ in $\mathcal{H}_0$), then $y$ has rank $j$, where $j$ is maximal s.t.* $\textsc{mass}(y)/\textsc{norm}(x) \geq \lambda_j$.

(c) *Let $y$ be a child of a rank $j$ node. We call $y$ stunted if* $\textsc{mass}(y)/\textsc{norm}(x) < \lambda_{j-1}/2$. *Each node has at most one stunted child.*

(d) *Let $y$ be of rank $j$. The children of $y$ can be divided into three sets: $Y_1$, $Y_2$, and a singleton $\{z\}$ such that* $(\textsc{mass}(Y_1) + \textsc{mass}(Y_2))/\textsc{norm}(x) < (2 + o(1)) \cdot \lambda_j$.

(e) *Let $\mathcal{X}$ be the nodes of $H(x)$ of some specific rank. Then* $\sum_{y \in \mathcal{X}} \textsc{mass}(y) \leq 2 \cdot \textsc{mass}(x)$.

Before moving on, let us examine some features of Property 5.1. Part (a) is asserted to guarantee that $\mathcal{H}$ is proper. Part (b) shows how we set the rank of leaves of $H(x)$. Part (c) says that at most one child of any node is less than half its ideal mass. Part (d) is a little technical but basically says that for a rank $j$ node $y$, although $\textsc{mass}(y)$ may be huge, the children of $y$ can be divided into sets $Y_1, Y_2, \{z\}$ such that $Y_1$ and $Y_2$ are of reasonable mass—around $\lambda_j \cdot \textsc{norm}(x)$. However, no bound is placed on the mass contributed by $z$. Part (e) says that if we restrict our attention to the nodes of a particular rank, their subgraphs do not overlap in too many places. To see how two subgraphs might overlap, consider $\{x_i\}$, the set of nodes of some rank in $H(x)$. By our construction it will always be the case that the vertex sets $\{V(x_i)\}$ are disjoint; however, this does not imply that the subtrees $\{M(x_i)\}$ are edge-disjoint because $M(x_i)$ can, in general, be much larger than $V(x_i)$.

We show in section 5.5 that if $\mathcal{H}$ is a refinement of $\mathcal{H}_0$ and $\mathcal{H}$ satisfies Property 5.1, then $\phi(\mathcal{H}) = O(n)$ and $\psi(\mathcal{H}) = O(n\log^* n)$. Recall from Lemmas 4.4 and 4.5 that $\phi(\mathcal{H})$ and $\psi(\mathcal{H})$ are terms in the running times of VISIT and VISIT-B, respectively.

**5.2. Phase 1: The MST and the coarse hierarchy.** Pettie and Ramachandran [PR02c] recently gave an MST algorithm that runs in time proportional to the decision-tree complexity of the MST problem. As the complexity of MST is trivially $\Omega(m)$ and only known to be $O(m\alpha(m, n))$ [Chaz00], it is unknown whether this cost will dominate or be dominated by the $\textsc{split-findmin}(m, n)$ term. (This issue is mainly of theoretical interest.) In the analysis we use $\textsc{mst}(m, n)$ to denote the cost of computing the MST. This may be interpreted as the decision-tree complexity of MST [PR02c] or the randomized complexity of MST, which is known to be linear [KKT95, PR02b].

Recall from section 5.1.1 that $\mathcal{H}_0$ was defined w.r.t. an arbitrary increasing sequence of $\textsc{norm}$-values. We describe below exactly how the $\textsc{norm}$-values are chosen, then prove that $\mathcal{H}_0$ is a proper hierarchy. Our method depends on how large $r$ is, which is the ratio of the maximum-to-minimum edge length in the minimum spanning tree. If $r < 2^n$, which can easily be determined in $O(n)$ time, then the possible $\textsc{norm}$-values are $\{\ell_{\min} \cdot 2^i \;:\; 0 \leq i \leq \log r + 1\}$, where $\ell_{\min}$ is the minimum edge length in the graph. If $r \geq 2^n$, then let $e_1, \ldots, e_{n-1}$ be the edges in $M$ in nondecreasing order by length and let $J = \{1\} \cup \{j : \ell(e_j) > n \cdot \ell(e_{j-1})\}$ be the indices that mark large separations in the $(\ell(e_i))_{1\leq i<n}$ sequence. The possible NORM-values are then $\{\ell(e_j) \cdot 2^i \ : \ i \geq 0,\ j \in J \text{ and } \ell(e_j) \cdot 2^i < \ell(e_{j+1})\}$.

Under either definition, $\textsc{norm}_i$ is the $i$th largest NORM-value, and for an edge $e \in E(M)$, $\textsc{norm}(e) = \textsc{norm}_i$ if $\ell(e) \in [\textsc{norm}_i, \textsc{norm}_{i+1})$. Notice that if no edge length falls within the interval $[\textsc{norm}_i, \textsc{norm}_{i+1})$, then $\textsc{norm}_i$ is an unused NORM-value. We only need to keep track of the NORM-values in use, of which there are no more than $n-1$.

LEMMA 5.2. *The coarse hierarchy $\mathcal{H}_0$ is a proper hierarchy.*

*Proof.* As we observed before, parts (i), (iii), and (iv) of Definition 4.1 are satisfied for any monotonically increasing sequence of NORM-values. Definition 4.1(ii) states that if $x$ is a hierarchy node, either $\textsc{norm}(p(x))/\textsc{norm}(x)$ is an integer or $\textsc{diam}(x)/\textsc{norm}(p(x)) < 1$. Suppose that $x$ is a hierarchy node and $\textsc{norm}(p(x))/\textsc{norm}(x)$ is not integral; then $\textsc{norm}(x) = \ell(e_{j_1}) \cdot 2^{i_1}$ and $\textsc{norm}(p(x)) = \ell(e_{j_2}) \cdot 2^{i_2}$, where $j_2 > j_1$. By our method for choosing the NORM-values, the lengths of all MST edges are either at least $\ell(e_{j_2})$ or less than $\ell(e_{j_2})/n$. Since edges in $M(x)$ have length less than $\ell(e_{j_2})$, and hence less than $\ell(e_{j_2})/n$, $\textsc{diam}(x) < (|V(x)|-1) \cdot \ell(e_{j_2})/n < \ell(e_{j_2}) \leq \textsc{norm}(p(x))$. ∎

LEMMA 5.3. *We can compute the minimum spanning tree $M$, and* $\textsc{norm}(e)$ *for all $e \in E(M)$, in $O(\textsc{mst}(m,n) + \min\{n \log\log r,\ n \log n\})$ time.*

*Proof.* $\textsc{mst}(m,n)$ represents the time to find $M$. If $r < 2^n$, then by Lemma 2.2 we can find $\textsc{norm}(e)$ for all $e \in M$ in $O(\log r + n \log\log r) = O(n \log\log r)$ time. If $r \geq 2^n$, then $n \log\log r = \Omega(n \log n)$, so we simply sort the edges of $M$ and determine the indices $J$ in $O(n \log n)$ time. Suppose there are $n_j$ edges $e$ s.t. $\textsc{norm}(e)$ is of the form $\ell(e_j) \cdot 2^i$. Since $\ell(e)/\ell(e_j) \leq n^{n_j}$, we need only generate $n_j \log n$ values of the form $\ell(e_j) \cdot 2^i$. A list of the $\sum_j n_j \log n = n \log n$ possible NORM-values can easily be generated in sorted order. By merging this list with the list of MST edge lengths, we can determine $\textsc{norm}(e)$ for all $e \in M$ in $O(n \log n)$ time. ∎

Lemma 5.4, given below, will come in handy in bounding the running time of our preprocessing and SSSP algorithms. It says that the total normalized mass in $\mathcal{H}_0$ is linear in $n$. Variations of Lemma 5.4 are at the core of the hierarchy approach [Tho99, Hag00, Pet04, Pet02b].

LEMMA 5.4.

$$\sum_{x \in \mathcal{H}_0} \frac{\textsc{mass}(x)}{\textsc{norm}(x)} < 4(n-1).$$

*Proof.* Recall that the notation $\textsc{norm}(e) = \textsc{norm}_i$ stands for $\ell(e) \in [\textsc{norm}_i, \textsc{norm}_{i+1})$, where $\textsc{norm}_i$, is the $i$th largest NORM-value. Observe that if $e \in M$ is an MST edge with $\textsc{norm}(e) = \textsc{norm}_i$, $e$ can be included in $\textsc{mass}(x)$ for no more than one $x$ at levels $i, i+1, \ldots$ in $\mathcal{H}_0$. Also, it follows from our definitions that for every $\textsc{norm}_i$ in use, $\textsc{norm}_{i+1}/\textsc{norm}_i \geq 2$, and for any MST edge, $\ell(e)/\textsc{norm}(e) < 2$. Therefore, we can bound the normalized mass in $\mathcal{H}_0$ as

$$\begin{aligned}\sum_{x \in \mathcal{H}_0} \frac{\textsc{mass}(x)}{\textsc{norm}(x)} &\leq \sum_{\substack{e \in M \\ \textsc{norm}(e)=\textsc{norm}_i}} \sum_{j=i}^{\infty} \frac{\ell(e)}{\textsc{norm}_j} \\ &\leq \sum_{\substack{e \in M \\ \textsc{norm}(e)=\textsc{norm}_i}} \sum_{j=i}^{\infty} \frac{\ell(e)}{2^{j-i} \cdot \textsc{norm}_i} < 4(n-1). \qquad \blacksquare\end{aligned}$$

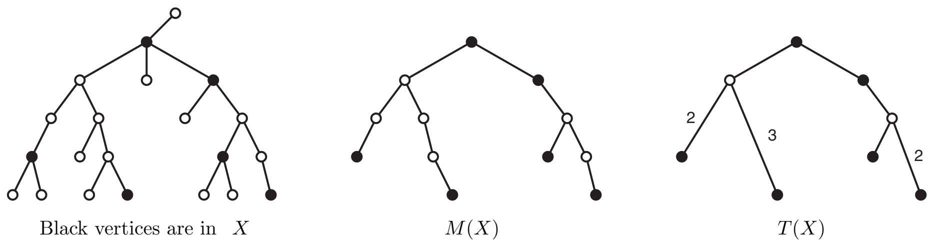


FIG. 2. *On the left is a subtree of $M$, the MST, where $X$ is the set of blackened vertices. In the center is $M(X)$, the minimal subtree of $M$ connecting $X$, and on the right is $T(X)$, derived from $M(X)$ by splicing out unblackened degree 2 nodes in $M(X)$ and adjusting edge lengths appropriately. Unless otherwise marked, all edges in this example are of length* 1.

Implicit in Lemma 5.4 is a simple accounting scheme where we treat mass, or more accurately normalized mass, as a currency equivalent to computational work. A hierarchy node $x$ "owns" $\text{MASS}(x)/\text{NORM}(x)$ units of currency. If we can then show that the share of some computation relating to $x$ is bounded by $k$ times its currency, the total time for this computation is $O(kn)$, that is, of course, if all computation is attributable to some hierarchy node. Although simple, this accounting scheme is very powerful and can become quite involved [Pet04, Pet02b, Pet03].

**5.3. Phase 2: Constructing $\boldsymbol{T(x)}$ trees.** Although it is possible to construct an $H(x)$ that satisfies Property 5.1 by working directly with the subtree $M(x)$, we are unable to efficiently compute $H(x)$ in this way. The problem is that we have time roughly proportional to the size of $H(x)$ to construct $H(x)$, whereas $M(x)$ could be significantly larger than $H(x)$. Our solution is to construct a *succinct tree* $T(x)$ that preserves the essential structure of $M(x)$ while having size roughly the same as $H(x)$.

For $X \subseteq V(G)$, let $T(X)$ be the subtree derived from $M(X)$ by *splicing* out all single-child vertices in $V(M(X)) - X$. That is, we replace each chain of vertices in $M(X)$, where only the end vertices are potentially in $X$, with a single edge; the length of this edge is the sum of its corresponding edge lengths in $M(X)$. Since there is a correspondence between vertices in $T(X)$ and $M$, we will refer to $T(X)$ vertices by their names in $M$. Figure 2 gives examples of $M(X)$ and $T(X)$ trees, where $X$ is the set of blackened vertices.

If $x \in \mathcal{H}_0$ and $\{x_j\}_j$ is the set of children of $x$, then let $T(x)$ be the tree $T(\{\text{ROOT}(M(x_j))\}_j)$; note that $\text{ROOT}(M(x))$ is included in $\{\text{ROOT}(M(x_j))\}_j$. Since only some of the edges of $M(x)$ are represented in $T(x)$, it is possible that the total length of $T(x)$ is significantly less than the total length of $M(x)$ (the MASS of $M(x)$); however, we will require that any subgraph of $T(x)$ have roughly the same mass as an equivalent subgraph in $M(x)$. In order to accomplish this we attribute certain amounts of mass to the *vertices* of $T(x)$ as follows. Suppose that $y$ is a child of $x$ in $\mathcal{H}_0$ and $v = \text{ROOT}(y)$ is the corresponding root vertex in $T(x)$. We let $\text{MASS}(v) = \text{MASS}(y)$. All other vertices in $T(x)$ have zero mass. The mass of a subtree of $T(x)$ is then the sum of its edge lengths plus the collective mass of its vertices.

We will think of a subtree of $T(x)$ as corresponding to a subtree of $M(x)$. Each edge in $T(x)$ corresponds naturally to a path in $M(x)$, and each vertex in $T(x)$ with nonzero mass corresponds to a subtree of $M(x)$.

LEMMA 5.5. *For $x \in \mathcal{H}_0$,*

(i) $\text{DEG}(x) \leq |V(T(x))| < 2 \cdot \text{DEG}(x)$;

(ii) *let $T_1$ be a subtree of $T(x)$ and $T_2$ be the corresponding tree in $M(x)$. Then*

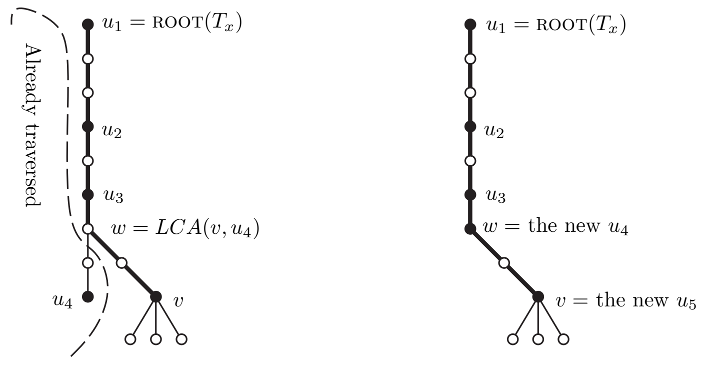


FIG. 3. *The blackened vertices represent those known to be in $T(x)$. The active path of the traversal is shown in bold edges. Before $v$ is processed (left) the stack consists of $\langle u_1, u_2, u_3, u_4\rangle$, where $u_4$ is the last vertex in the traversal known to be in $T(x)$ and $w = LCA(v, u_4)$, which implies $w \in T(x)$. After $v$ is processed (right) the stack is set to $\langle u_1, u_2, u_3, w, v\rangle$ and $w$ is blackened.*

$\text{MASS}(T_2) \le \text{MASS}(T_1) \le 2 \cdot \text{MASS}(T_2)$.

*Proof.* Part (i) follows from two observations. First, $T(x)$ has no degree two vertices. Second, there are at most $\text{DEG}(x)$ leaves of $T(x)$ since each such leaf corresponds to a vertex $\text{ROOT}(M(y))$ for some child $y$ of $x$ in $\mathcal{H}_0$. Part (ii) follows since all mass in $T_2$ is represented in $T_1$, and each edge in $T_2$ contributes to the mass of at most one edge and one vertex in $T_1$. ☐

We construct $T(x)$ with a kind of depth first traversal of the minimum spanning tree, using the procedure SUCCINCT-TREE, given below. SUCCINCT-TREE focuses on some fixed $\mathcal{H}_0$-node $x$. We will explain how SUCCINCT-TREE works with the aid of the diagram in Figure 3. At every point in the traversal we maintain a stack of vertices $\langle u_1, \ldots, u_q\rangle$ consisting of all vertices known to be in $T(x)$, whose parents in $T(x)$ are not yet fixed. The stack has the following properties: $u_i$ is ancestral to $u_{i+1}$, $\langle u_1, \ldots, u_{q-1}\rangle$ are on the *active path* of the traversal, and $u_q$ is the last vertex known to be in $T(x)$ encountered in the traversal.

In Figure 3 the stack consists of $\langle u_1, \ldots, u_4\rangle$, where $\langle u_1, u_2, u_3\rangle$ are on the active path of the traversal, marked in bold edges. The preprocessing of $v$ (before making recursive calls) is to do nothing if $v \notin \{\text{ROOT}(M(x_j))\}_j$. Otherwise, we update the stack to reflect our new knowledge about the edges and vertices of $T(x)$. The vertex $w = LCA(u_q, v) = LCA(u_4, v)$ must be in $T(x)$. There are three cases: either $w$ is the ultimate or penultimate vertex in the stack ($u_q$ or $u_{q-1}$), that is, we already know $w \in T(x)$, or $w$ lies somewhere on the path between $u_q$ and $u_{q-1}$. Figure 3 diagrams the third situation. Because no $T(x)$ vertices were encountered in the traversal between $u_q = u_4$ and $v$, there can be no new $T(x)$ vertices discovered on the path between $u_q$ and $w$. Therefore, we can pop $u_q$ off the stack, designating its parent in $T(x)$ to be $w$, and push $w$ and $v$ onto the stack. The other two situations, when $w = u_q$ or $w = u_{q-1}$, are simpler. If $w = u_q$, then we simply push $v$ onto the stack, and if $w = u_{q-1}$, we pop $u_q$ off the stack and push $v$ on. Now consider the postprocessing of $v$ (performed after all recursive calls), and let $u_{q-1}, u_q$ be the last two vertices in the stack. Suppose that $v = u_{q-1}$. We cannot simply do nothing, because when the active path retracts there will be two stack vertices ($v = u_{q-1}$ and $u_q$) outside of the active path, contrary to the stack properties. However, because no $T(x)$ vertices were discovered between $u_q$ and $u_{q-1}$, we can safely say that $u_{q-1}$ is the parent of $u_q$ in $T(x)$. So, to maintain the stack properties, we pop off $u_q$ and add the edge $(u_q, u_{q-1})$ to $T(x)$.

```
SUCCINCT-TREE($v$): A procedure for constructing $T(x)$, for a given $x \in \mathcal{H}_0$.
        The argument $v$ is a vertex in the MST $M$.
        The stack for $T(x)$ consists of vertices $\langle u_1, \ldots, u_q \rangle$, which
        are known to be in $T(x)$ but whose parents in $T(x)$ are
        not yet known. All but $u_q$ are on the active path of the
        DFS traversal. Initially the stack for $T(x)$ is empty.

1.    If $v = \text{ROOT}(y)$, where $y$ is a child of $x$ in $\mathcal{H}_0$, then
2.        Let $w = LCA(v, u_q)$
3.        If $w \neq u_q$
4.            POP $u_q$ off the stack
5.            Designate $(u_q, w)$ an edge in $T(x)$
6.            If $w \neq u_{q-1}$
7.                PUSH $w$ on the stack
8.        PUSH $v$ on the stack
9.    Call SUCCINCT-TREE on all the children of $v$ in $M$
10.   Let $u_{q-1}, u_q$ refer to the last two elements in the current stack
11.   If $v = u_{q-1}$
12.       POP $u_q$ off the stack
13.       Designate $(u_q, u_{q-1}) = (u_q, v)$ an edge in $T(x)$
```

LEMMA 5.6. *Given the MST and a list of its edges ordered by level, $\mathcal{H}_0$ and $\{T(x)\}_x$ can be constructed in $O(n)$ time.*

*Proof.* We construct $\mathcal{H}_0$ with a union-find structure and mark all vertices in $M$ as roots of $M(x)$ for (one or more) $x \in \mathcal{H}_0$. It is easy to see that we can construct all $T(x)$ for $x \in \mathcal{H}_0$, with one tree traversal given in procedure SUCCINCT-TREE. We simply maintain a different stack for each $T_x$ under construction. Thus if $v$ is the root of several $M(y_1), M(y_2), \ldots$, where $y_i \in \mathcal{H}_0$, we simply reexecute lines 1–8 and 10–13 of SUCCINCT-TREE for each of $v$'s roles. Using a well-known union-find–based least common ancestors (LCA) algorithm [AHU76, Tar79b], we can compute the LCAs in line 2 in $O(n\alpha(n))$ time, since the number of finds is linear in the number of nodes in $\mathcal{H}_0$. If we use the scheme of Buchsbaum et al. [BKRW98] instead, the cost of finding LCAs is linear; however, since this algorithm is offline (it does not handle LCA queries in the middle of a tree traversal, unlike [AHU76, Tar79b]), we would need to determine what the LCA queries are with an initial pass over the tree. Finally, we compute the length function in $T(x)$ as follows. If $(u, v) \in E(T(x))$ and $v$ is ancestral to $u$ in $M$, then $\ell(u, v) = d_M(u, \text{ROOT}(M)) - d_M(v, \text{ROOT}(M))$, where $d_M$ is the distance function for $M$. Clearly the $d_M(\cdot, \text{ROOT}(M))$ function can be computed in $O(n)$ time. See Lemma 2.1 for a simulation of subtraction in the comparison-addition model. ∎

## 5.4. Phase 3: Constructing the refined hierarchy.

We show in this section how to construct an $H(x)$ from $T(x)$ that is consistent with Property 5.1.

The REFINE-HIERARCHY procedure, given as pseudocode below, constructs $H(x)$ in a bottom-up fashion by traversing the tree $T(x)$. A call to REFINE-HIERARCHY($v$), where $v \in T(x)$, will produce an array of sets $v[\cdot]$ whose elements are nodes in $H(x)$ that represent (collectively) the subtree of $T(x)$ rooted at $v$. The set $v[j]$ holds rank $j$ nodes, which, taken together, are not yet massive enough to become a rank $j+1$ node. We extend the mass notation to sets $v[\cdot]$ as follows. Bear in mind that this mass is w.r.t. the tree $T(x)$, not $M(x)$. By Lemma 5.5(ii), mass w.r.t. $T(x)$ is a good approximation to the mass of the equivalent subtree in $M(x)$:

$$\textsc{mass}(v[j]) \;=\; \textsc{mass}\left(\bigcup_{j' \le j} \; \bigcup_{y \in v[j']} V(y)\right).$$

```
REFINE-HIERARCHY($v$): Constructing $H(x)$, for a given $x \in \mathcal{H}_0$,
        where $v$ is a vertex in $T(x)$.
1.      Initialize $v[j] := \emptyset$ for all $j$.
2.      If $v = \textsc{root}(M(y))$ for some child $y$ of $x$ in $\mathcal{H}_0$
3.              Let $j$ be maximal s.t. $\textsc{mass}(y)/\textsc{norm}(x) \ge \lambda_j$
4.              $v[j] := \{y\}$    (i.e., $y$ is implicitly designated a rank $j$ node)
5.      For each child $w$ of $v$ in $T(x)$:
6.              REFINE-HIERARCHY($w$)
7.              For all $i$,  $v[i] := v[i] \cup w[i]$
8.              Let $j$ be maximal such that $\textsc{mass}(v[j])/\textsc{norm}(x) \ge \lambda_{j+1}$
9.              Promote $v[0], \ldots, v[j]$      (see Definition 5.7)
10.     If $v$ is the root of $T(x)$, promote $v[0], v[1], \ldots$ until one node remains.
        (This final node is the root of $H(x)$.)
```

The structure of Refine-Hierarchy is fairly simple. To begin with, we initialize $v[\cdot]$ to be an array of empty sets. Then, if $v$ is a root vertex of a child $y$ of $x$ in $\mathcal{H}_0$, we create a node representing $y$ and put it in the proper set in $v[\cdot]$; which set receives $y$ depends only on $\textsc{mass}(y)$. Next we process the children of $v$. On each pass through the loop, we pick an as yet unprocessed child $w$ of $v$; recurse on $w$, producing sets $w[\cdot]$ representing the subtree rooted at $w$; and then merge the sets $w[\cdot]$ into their counterparts in $v[\cdot]$. At this point, the mass of some sets may be beyond a critical threshold: the threshold for $v[j]$ is $\lambda_{j+1} \cdot \textsc{norm}(x)$. In order to restore a quiescent state in the sets $v[\cdot]$ we perform *promotions* until no set's mass is above threshold.

Definition 5.7. *Promoting the set $v[j]$ involves removing the nodes from $v[j]$, making them the children of a new rank $j+1$ node, and then placing this node in $v[j+1]$. There is one exception: if $|v[j]| = 1$, then to comply with Definition* 4.1(iii), *we simply move the node from $v[j]$ to $v[j+1]$. Promoting the sets $v[0], v[1], \ldots, v[j]$ means promoting $v[0]$, then $v[1]$, up to $v[j]$, in that order.*

Suppose that after merging $w[\cdot]$ into $v[\cdot]$, $j$ is maximal such that $\textsc{mass}(v[j])$ is beyond its threshold of $\lambda_{j+1} \cdot \textsc{norm}(x)$ (there need not be such a $j$). We promote the sets $v[0], \ldots, v[j]$, which has the effect of emptying $v[0], \ldots, v[j]$ and adding a new node to $v[j+1]$ representing the nodes formerly in $v[0], \ldots, v[j]$. Lemma 5.8, given below, shows that we can compute the $H(x)$ trees in linear time.

Lemma 5.8. *Given $\{T(x)\}_x$, $\{H(x)\}_x$ can be constructed to satisfy Property* 5.1 *in $O(n)$ time.*

*Proof.* We first argue that Refine-Hierarchy produces a refinement $\mathcal{H}$ of $\mathcal{H}_0$ that satisfies Property 5.1. We then look at how to implement it in linear time.

Property 5.1(a) states that internal nodes in $H(x)$ must have $\textsc{norm}$-values equal to that of $x$, which we satisfy by simply assigning them the proper $\textsc{norm}$-values, and that no node of $H(x)$ has one child. By our treatment of one-element sets in the promotion procedure of Definition 5.7, it is simply impossible to create a one-child node in $H(x)$. Property 5.1(e) follows from Lemma 5.5(ii) and the observation that the mass (in $T(x)$) represented by nodes of the same rank is disjoint. Now consider Property 5.1(c), regarding stunted nodes. We show that whenever a set $v[j]$ accepts a new node $z$, either $v[j]$ is immediately promoted, or $z$ is not stunted, or the promotion of $z$ into $v[j]$ represents the *last* promotion in the construction of $H(x)$. Consider the pattern of promotions in line 9. We promote the sets $v[0], \ldots, v[j]$ in a cascading fashion: $v[0]$ to $v[1]$, $v[1]$ to $v[2]$, and so on. The only set accepting a new node that is not immediately promoted is $v[j+1]$, so in order to prove Property 5.1(c) we must show that the node derived from promoting $v[0], \ldots, v[j]$ is not stunted. By choice of $j$, $\textsc{mass}(v[j]) \geq \lambda_{j+1} \cdot \textsc{norm}(x)$, where mass is w.r.t. the tree $T(x)$. By Lemma 5.5(ii) the mass of the equivalent tree in $M(x)$ is at least $\lambda_{j+1} \cdot \textsc{norm}(x)/2$, which is exactly the threshold for this node being stunted. Finally, consider Property 5.1(d). Before the merging step in line 7, none of the sets in $v[\cdot]$ or $w[\cdot]$ is massive enough to be promoted. Let $v[\cdot]$ and $w[\cdot]$ denote the sets associated with $v$ and $w$ before the merging in step 7, and let $v'[\cdot]$ denote the set associated with $v$ after step 7. By the definition of $\textsc{mass}$ we have

$$\textsc{mass}(v'[j]) = \textsc{mass}(v[j]) + \textsc{mass}(w[j]) + \ell(v,w) < 2 \cdot \lambda_{j+1} \cdot \textsc{norm}(x) + \ell(v,w).$$

Since $(v,w)$ is an edge in $T(x)$, it can be arbitrarily large compared to $\textsc{norm}(x)$, meaning we cannot place any reasonable bound on $\textsc{mass}(v'[j])$ after the merging step. Let us consider how Property 5.1(d) is maintained. Suppose that $v'[j]$ is promoted in lines 9 or 10, and let $y$ be the resulting rank $j+1$ node. Using the terminology from Property 5.1(d), let $Y_1 = v[j], Y_2 = w[j]$ and let $z$ be the node derived by promoting $v'[0], \ldots, v'[j-1]$. Since neither $v[j]$ nor $w[j]$ was sufficiently massive to be promoted before they were merged, we have $(\textsc{mass}(Y_1) + \textsc{mass}(Y_2))/\textsc{norm}(x) < 2\lambda_{j+1}$. This is slightly stronger than what Property 5.1(d) calls for, which is the inequality $< (2 + o(1))\lambda_{j+1}$. We'll see why the $(2 + o(1))$ is needed below.

Suppose that we implemented Refine-Hierarchy in a straightforward manner. Let $L$ be the (known) maximum possible index of any nonempty set $v[\cdot]$ during the course of Refine-Hierarchy. One can easily see that the initialization in lines 1–4 takes $O(L+1)$ time and that, exclusive of recursive calls, each time through the for-loop in line 5 takes $O(L+1)$ amortized time. (The bound on line 5 is *amortized* since promoting a set $v[j]$ takes worst case $O(|v[j]|+1)$ time but only constant amortized time.) The only hidden costs in this procedure are updating the $\textsc{mass}$ of sets, which is done as follows. After the merging step in line 7, we simply set $\textsc{mass}(v[j]) := \textsc{mass}(v[j]) + \ell(v,w) + \textsc{mass}(w[j])$ for each $j \leq L$. Therefore the total cost of computing $H(x)$ from $T(x)$ is $O((L+1) \cdot |T(x)|)$. We can bound $L$ as $L \leq 2\log^*(4n)$ as follows. The first node placed in any previously empty set is unstunted; therefore, by Lemma 5.1, the maximum nonempty set has rank at most $2\log^*(\textsc{mass}(T(x))/\textsc{norm}(x))$. By Lemma 5.5(ii) and the construction of $\mathcal{H}_0$, $\textsc{mass}(T(x)) \leq 2 \cdot \textsc{mass}(M(x)) < 4(n-1) \cdot \textsc{norm}(x)$.

In order to reduce the cost to linear we make a couple adjustments to the Refine-Hierarchy procedure. First, $v[\cdot]$ is represented as a linked list of nonempty sets. Second, we update the mass variables in a lazy fashion. The time for steps 1–4 is dominated by the time to find the appropriate $j$ in step 3, which takes time $t_1$—see below. The time for merging the $v[\cdot]$ and $w[\cdot]$ sets in line 7 is only proportional to the shorter list; this time bound is given by expression $t_2$ below.

$$t_1 = O\left(1 + \log^* \frac{\textsc{mass}(v)}{\textsc{norm}(x)}\right),$$

$$t_2 = O\left(1 + \log^* \frac{\min\{\textsc{mass}(v[\cdot]), \textsc{mass}(w[\cdot])\}}{\textsc{norm}(x)}\right),$$

where $\textsc{mass}(v[\cdot])$ is just the total mass represented by the $v[\cdot]$ sets. We update the mass of only the first $t_1 + t_2$ sets in $v[\cdot]$, and, as a rule, we update $v[j+1]$ half as often as $v[j]$. It is routine to show that Refine-Hierarchy will have a lower bound on the mass of $v[j]$ that is off by a 1+o(1) factor, where the $o(1)$ is a function of $j$.[7] This leads to the conspicuous $2 + o(1)$ in Property 5.1(d). To bound the cost of Refine-Hierarchy we model its computation as a binary tree: leaves represent the creation of nodes in lines 1–4, and internal nodes represent the merging events in line 7. The cost of a leaf $f$ is $\log^*(\textsc{mass}(f)/\textsc{norm}(x))$, and the cost of an internal node $f$ with children $f_1$ and $f_2$ is $1 + \log^*(\min\{\textsc{mass}(f_1)/\textsc{norm}(x), \textsc{mass}(f_2)/\textsc{norm}(x)\})$. We can think of charging the cost of $f$ collectively to the mass in the subtree of $f_1$ or $f_2$, whichever is smaller. Therefore, no unit of mass can be charged for two nodes $f$ and $g$ if the total mass under $f$ is within twice the total mass under $g$. The total cost is then

$$\sum_f cost(f) = O\left(|T(x)| + \frac{\textsc{mass}(T(x))}{\textsc{norm}(x)} \cdot \sum_{i=0}^{\infty} \frac{\log^*(2^i)}{2^i}\right) = O\left(\frac{\textsc{mass}(x)}{\textsc{norm}(x)}\right).$$

The last equality follows because $|T(x)| = O(\textsc{mass}(T(x))/\textsc{norm}(x)) = O(\textsc{mass}(M(x))/\textsc{norm}(x))$. Summing over all $x \in \mathcal{H}_0$, the total cost of constructing $\{H(x)\}_{x\in\mathcal{H}_0}$ is, by Lemma 5.4, $O(n)$. ◻

Lemma 5.9. *In $O(\textsc{mst}(m,n) + \min\{n \log\log r, n \log n\})$ time we can construct both the coarse hierarchy $\mathcal{H}_0$ and a refinement $\mathcal{H}$ of $\mathcal{H}_0$ satisfying Property* 5.1.

*Proof.* The proof follows from Lemmas 5.3, 5.6, and 5.8. ◻

**5.5. Analysis.** In this section we prove bounds on the running times of Visit and Visit-B, given an appropriate refined hierarchy such as the one constructed in section 5.4. Theorem 1.1 follows directly from Lemma 5.10, given below, and Lemma 5.9.

Lemma 5.10. *Let $\mathcal{H}$ be any refinement of $\mathcal{H}_0$ satisfying Property* 5.1*. Using $\mathcal{H}$,* Visit *computes SSSP in $O(\textsc{split-findmin}(m,n))$ time, and* Visit-B *computes SSSP in $O(m + n\log^* n)$ time.*

*Proof.* We prove that $\phi(\mathcal{H}) = O(n)$ and $\psi(\mathcal{H}) = O(n\log^* n)$. Together with Lemmas 4.4 and 4.5, this will complete the proof.

With the observation that $\textsc{mass}(x)$ is an upper bound on the diameter of $V(x)$, we will substitute $\textsc{mass}(x)$ for $\textsc{diam}(x)$ in the functions $\phi$ and $\psi$. By Lemma 5.4, the first sum in $\phi$ is $O(n)$. The first sum of $\psi(\mathcal{H})$ is much like that in $\phi$, except we sum over *all* nodes in $\mathcal{H}$, not just those nodes that also appear in $\mathcal{H}_0$. By Property 5.1(a), (c), and (d) and Lemma 5.1, the maximum rank of any node in $H(x)$ is

[7]The proof of this is somewhat tedious. Basically one shows that for $i < j$ the mass of $v[i]$ can be updated at most $2^{j-i} - 1$ times before the mass of $v[j]$ is updated. Since $2^{j-1} - 1 \cdot \lambda_i \ll \lambda_j$, our neglecting to update the mass of $v[j]$ causes a negligible error.

$2\log^*(\textsc{mass}(x)/\textsc{norm}(x)) \leq 2\log^* n$. By Property 5.1(e) the total mass of nodes of one rank in $H(x)$ is bounded by $2 \cdot \textsc{mass}(x)$. Therefore, we can bound the first sum in $\psi(\mathcal{H})$ as $\sum_x \textsc{mass}(x)/\textsc{norm}(x) \leq 4\log^* n \cdot \sum_{x \in \mathcal{H}_0} \textsc{mass}(x)/\textsc{norm}(x)$, which is $O(n\log^* n)$ by Lemma 5.4.

We now turn to the second summations in $\phi(\mathcal{H})$ and $\psi(\mathcal{H})$, which can be written as $\sum_x \textsc{deg}(x) \log(\textsc{mass}(x)/\textsc{norm}(x))$ and $\sum_x \textsc{deg}(x) \log \textsc{deg}(x)$, respectively. Since $\textsc{deg}(x) \leq 1 + \textsc{mass}(x)/\textsc{norm}(x)$, any bound established on the first summation will extend to the second.

Let $y$ be a rank $j$ node. Using the terms from Property 5.1(d), let $\alpha = (\textsc{mass}(Y_1)+ \textsc{mass}(Y_2))/\textsc{norm}(y)$ and $\beta = \textsc{mass}(y)/\textsc{norm}(y) - \alpha$. Property 5.1(c), (d) imply that $\alpha < (2 + o(1)) \cdot \lambda_j$ and that $\textsc{deg}(y) \leq 2\alpha/\lambda_{j-1} + 2$, where the $+2$ represents the stunted child and the child $z$ exempted from Property 5.1(d):

$$\begin{aligned}
\textsc{deg}(y) \log \frac{\textsc{mass}(y)}{\textsc{norm}(y)} &\leq \left(\frac{2\alpha}{\lambda_{j-1}} + 2\right) \log(\alpha + \beta) \qquad \{\text{see explanations below}\} \\
&= O\left(\frac{\max\{\alpha \log(2\lambda_j), \beta\}}{\lambda_{j-1}}\right) \\
&= O\left(\frac{\alpha + \beta}{2^{j-1}}\right) \;=\; O\left(\frac{\textsc{mass}(y)}{\textsc{norm}(y) \cdot 2^{j-1}}\right).
\end{aligned}$$

The first line follows from our bound on $\textsc{deg}(y)$ and the definitions of $\alpha$ and $\beta$. The second line follows since $\alpha < (2+o(1))\lambda_j$ and $\alpha \log(\alpha+\beta) = O(\max\{\alpha \log \alpha, \beta\})$. The last line follows since $\log \lambda_j = \lambda_{j-1}/2^{j-1} > 1$. By the above bound and Property 5.1(e), $\sum_{y \in H(x)} \textsc{deg}(y) \log(\textsc{mass}(y)/\textsc{norm}(y)) = O(\textsc{mass}(x)/\textsc{norm}(x))$. Therefore, by Lemma 5.4, the second summations in both $\phi(\mathcal{H})$ and $\psi(\mathcal{H})$ are bounded by $O(n)$. ☐

## 6. Limits of hierarchy-type algorithms.

In this section we state a simple property (Property 6.1) of all hierarchy-type algorithms and give a lower bound on any undirected SSSP algorithm satisfying that property. The upshot is that our SSSP algorithm is optimal (up to an inverse-Ackermann factor) for a fairly large class of SSSP algorithms, which includes all hierarchy-type algorithms, variations on Dijkstra's algorithm, and even a heuristic SSSP algorithm [G01].

We will state Property 6.1 in terms of directed graphs. Let $\textsc{cycles}(u, v)$ denote the set of all cycles, including nonsimple cycles, that pass through both $u$ and $v$, and let $\textsc{sep}(u, v) = \min_{C \in \textsc{cycles}(u,v)} \max_{e \in C} \ell(e)$. Note that in undirected graphs $\textsc{sep}(u, v)$ corresponds exactly to the longest edge on the MST path between $u$ and $v$.

Property 6.1. *An SSSP algorithm with the* hierarchy property *computes, aside from shortest paths, a permutation $\pi_s : V(G) \to V(G)$ such that for any vertices $u, v$, we find $d(s, u) \geq d(s, v) + \textsc{sep}(u, v) \implies \pi_s(u) > \pi_s(v)$, where $s$ is the source and $d$ the distance function.*

The permutation $\pi_s$ corresponds to the order in which vertices are visited when the source is $s$. Property 6.1 says that $\pi_s$ is loosely sorted by distance, but may invert pairs of vertices if their relative distance is less than their $\textsc{sep}$-value. To see that our hierarchy-based algorithm satisfies Property 6.1, consider two vertices $u$ and $v$. Let $x$ be the LCA of $u$ and $v$ in $\mathcal{H}$, and let $u'$ and $v'$ be the ancestors of $u$ and $v$, respectively, which are children of $x$. By our construction of $\mathcal{H}$, $\textsc{norm}(x) \leq \textsc{sep}(u, v)$. If $d(s, u) \geq d(s, v) + \textsc{sep}(u, v)$, then $d(s, u) \geq d(s, v) + \textsc{norm}(x)$, and therefore the recursive calls on $u'$ and $v'$ that cause $u$ and $v$ to be visited are not passed the same interval argument, since both intervals have width $\textsc{norm}(x)$. The recursive call on $u'$

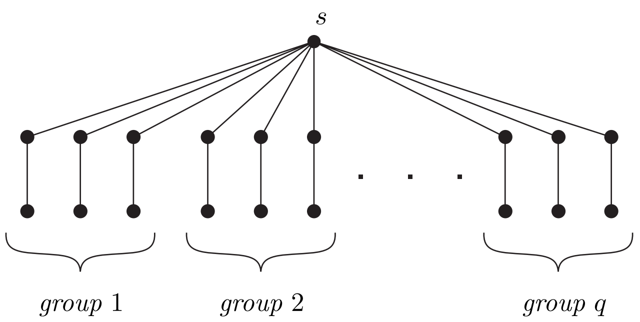


FIG. 4. *The minimum spanning tree of the graph.*

must, therefore, precede the recursive call on $v'$, and $u$ must be visited before $v$.

THEOREM 6.1. *Suppose that our computational model allows any set of functions from $\mathbb{R}^{O(1)} \to \mathbb{R}$ and comparison between two reals. Any SSSP algorithm for real-weighted graphs satisfying Property* 6.1 *makes $\Omega(m + \min\{n \log\log r,\, n \log n\})$ operations in the worst case, where $r$ is the ratio of the maximum to minimum edge length.*

*Proof.* Let $q$ be an integer. Assume without loss of generality that $2q$ divides $n-1$. The MST of the input graph is as depicted in Figure 4. It consists of the source vertex $s$, which is connected to $p = (n-1)/2$ vertices in the top row, each of which is paired with one vertex in the bottom row. All the vertices (except $s$) are divided into disjoint *groups*, where group $i$ consists of exactly $p/q$ randomly chosen pairs of vertices. There are exactly $p!/(p/q)!^q = q^{\Omega(p)}$ possible group arrangements. We will show that any algorithm satisfying Property 6.1 must be able to distinguish them.

We choose edge lengths as follows. All edges in group $i$ have length $2^i$. This includes edges from $s$ to the group's top row and between the two rows. Other non-MST edges are chosen so that shortest paths from $s$ correspond to paths in the MST. Let $v_i$ denote any vertex in the bottom row of group $i$. Then $d(s, v_i) = 2 \cdot 2^i$ and $\text{SEP}(v_i, v_j) = 2^{max\{i,j\}}$. By Property 6.1, $v_i$ must be visited before $v_j$ if $d(s, v_i) + \text{SEP}(v_i, v_j) \leq d(s, v_j)$, which is true for $i < j$ since $2 \cdot 2^i + 2^j \leq 2 \cdot 2^j$. Therefore, any algorithm satisfying Property 6.1 must be prepared to visit vertices in $q^{\Omega(p)}$ distinct permutations and make at least $\Omega(p \log q) = \Omega(n \log\log r)$ comparisons in the worst case. It also must include every non-MST edge in at least one operation, which gives the lower bound. $\square$

Theorem 6.1 shows that our SSSP algorithm is optimal among hierarchy-type algorithms, to within a tiny inverse-Ackermann factor. A lower bound on *directed* SSSP algorithms satisfying Property 6.1 is given in [Pet04]. Theorem 6.1 differs from that lower bound in two respects. First, the [Pet04] bound is $\Omega(m + \min\{n \log r,\, n \log n\})$, which is $\Omega(m + n \log n)$ for even reasonably small values of $r$. Second, the [Pet04] bound holds even if the algorithm is allowed to compute the SEP function (and sort the values) for free. Contrast this with our SSSP algorithm, where the main obstacle to achieving linear time is the need to sort the SEP-values.

## 7. Discussion.

We have shown that with a near-linear time investment in preprocessing, SSSP queries can be answered in very close to linear time. Furthermore, among a natural class of SSSP algorithms captured by Property 5.1, our SSSP algorithm is optimal, aside from a tiny inverse-Ackermann factor. We can imagine several avenues for further research, the most interesting of which is developing a feasible alternative to Property 5.1 that does not have an intrinsic sorting bottleneck. This would be a backward approach to algorithm design: first we define a desirable property, then we hunt about for algorithms with that property. Another avenue, which might have some real-world impact, is to reduce the preprocessing cost of the *directed* shortest path algorithm in [Pet04] from $O(mn)$ to near-linear, as it is in our algorithm.

The marginal cost of computing SSSP with our algorithm may or may not be linear; it all depends on the complexity of the split-findmin structure. This data structure, invented first by Gabow [G85a] for use in a weighted matching algorithm, actually has connections with other fundamental problems. For instance, it can be used to solve both the minimum spanning tree and shortest path tree sensitivity analysis problems [Pet03]. (The sensitivity analysis problem is to decide how much each edge's length can be perturbed without changing the solution tree.) Therefore, by Theorem 4.2 both these problems have complexity $O(m \log \alpha(m,n))$, an $\alpha / \log \alpha$ improvement over Tarjan's path-compression–based algorithm [Tar82]. If we consider the *offline* version of the split-findmin problem, where all splits and decrease-keys are given in advance, one can show that it is reducible to *both* the MST problem and the MST sensitivity analysis problem. None of these reductions proves whether $\textsc{mst}(m,n)$ dominates $\textsc{split-findmin}(m,n)$ or vice versa; however, they do suggest that we have no hope of solving the MST problem [PR02b, PR02c] without first solving the manifestly simpler split-findmin and MST sensitivity analysis problems.

The experimental study of Pettie, Ramachandran, and Sridhar [PRS02] shows that our algorithm is very efficient in practice. However, the [PRS02] study did not explore all possible implementation choices, such as the proper heap to use, the best preprocessing algorithm, or different implementations of the split-findmin structure. To our knowledge no one has investigated whether the other hierarchy-type algorithms [Tho99, Hag00, Pet04] are competitive in real-world scenarios.

An outstanding research problem in parallel computing is to bound the time-work complexity of SSSP. There are several published algorithms on the subject [BTZ98, CMMS98, KS97, M02, TZ96], though none runs in worst-case polylogarithmic time using work comparable to Dijkstra's algorithm. There is clearly a lot of parallelism in the hierarchy-based algorithms. Whether this approach can be effectively parallelized is an intriguing question.

**Appendix A. The bucket-heap.** The bucket-heap structure consists of an array of buckets, where the $i$th bucket spans the interval $[\delta + i\mu, \delta + (i+1)\mu)$, for fixed reals $\delta$ and $\mu$. Logically speaking, a heap item with key $\kappa$ appears in the bucket whose interval spans $\kappa$. We are never concerned about the relative order of items within the same bucket.

*Proof of Lemma* 4.3. Our structure simulates the logical specification given earlier; it actually consists of *levels* of bucket arrays. The level zero buckets are the ones referred to in the bucket-heap's specification, and the level $i$ buckets preside over disjoint intervals of $2^i$ level zero buckets. The interval represented by a higher-level bucket is the union of its component level zero buckets. Only one bucket at each level is *active*: it is the first one that presides over no closed level zero buckets; see Figure 5. Suppose that an item $x$ should *logically* be in the level zero bucket $B$. We maintain the invariant that $x$ is either *descending* and in the lowest active bucket presiding over $B$, or *ascending* and in some active bucket presiding over level zero buckets before $B$.

To insert a node we put it in the first open level zero bucket and label it as ascending. This clearly satisfies the invariant. The result of a decrease-key depends

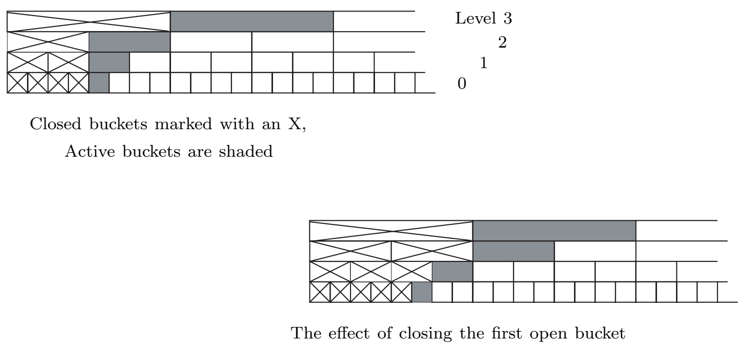

Level 3
2
1
0
Closed buckets marked with an X,
Active buckets are shaded
The effect of closing the first open bucket

FIG. 5. *Active buckets are shaded.*

on whether the node $x$ is ascending or descending. Suppose $x$ is ascending and in a bucket (at some level) spanning the interval $[a, b)$. If $\text{key}(x) < b$, we relabel it as descending; otherwise we do nothing. If $x$ is descending (or was just relabeled as descending), we move it to the lowest level active bucket consistent with the invariant. If $x$ drops $i \geq 0$ levels, we assume that this is accomplished in $O(i + 1)$ time; i.e., we search from its current level down, not from the bottom up.

Suppose that we close the first open level zero bucket $B$. According to the invariant all items that are logically in $B$ are descending and actually in $B$, so enumerating them is no problem; there will, in general, be ascending items in $B$ that do not logically belong there. In order to maintain the invariant we must deactivate all active buckets that preside over $B$ (including $B$). Consider one such bucket at level $i$. If $i > 0$, we move each descending node in it to the level $i - 1$ active bucket. For each ascending node (at level $i \geq 0$), depending on its key, we either move it to the level $i + 1$ active bucket and keep it ascending, or relabel it descending and move it to the proper active bucket at level $\leq i + 1$.

From the invariant it follows that no node $x$ appears in more than $2 \log \Delta_x + 1$ distinct buckets: $\log \Delta_x + 1$ buckets as an ascending node and another $\log \Delta_x$ as a descending node. Aside from this cost of moving nodes around, the other costs are clearly $O(N)$. ◻

We remark that the bucket-heap need not actually label the items. Whether an item is ascending or descending can be inferred from context.

**Appendix B. The split-findmin problem.** The split-findmin problem is to maintain a collection of sequences of weighted elements under the following operations:

| | |
|---|---|
| split($x$): | Split the sequence containing $x$ into two sequences: the elements up to and including $x$ and the rest. |
| decrease-key($x, \kappa$): | Set $\text{key}(x) = \min\{\text{key}(x), \kappa\}$. |
| findmin($x$): | Return the element in $x$'s sequence with minimum key. |

Gabow [G85a] gave an elegant algorithm for this problem that is nearly optimal. On an initial sequence of $n$ elements, it handles up to $n - 1$ splits and $m$ decrease-keys in $O((m + n)\alpha(m, n))$ time. Gabow's algorithm runs on a pointer machine [Tar79]. We now prove Theorem 4.2 from section 4.2.

*Proof of Theorem* 4.2. In Gabow's decrease-key routine a sequence of roughly $\alpha$ variables needs to be updated, although it is already known that their values are monotonically decreasing. We observe that, on a pointer machine, the same task can be accomplished in $O(\alpha)$ time using $O(\log\alpha)$ comparisons for a binary search. Using a simple two-level scheme, one can easily reduce the $n\alpha$ term in the running time to $n$. This gives the split-findmin algorithm that performs $O(m\log\alpha(m,n)+n)$ comparisons.

To get a potentially faster algorithm on the RAM model we construct all possible split-findmin solvers on inputs with at most $q=\log\log n$ elements and choose one that is close to optimal for all problem sizes. We then show how to compose this optimal split-findmin solver on $q$ elements with Gabow's structure to get an optimal solver on $n$ elements.

We consider only instances with $m'<q^2$ decrease-keys. If more decrease-keys are actually encountered, we can revert to Gabow's algorithm [G85a] or a trivial one that runs in $O(m')$ time.

We represent the state of the solver with three components: a bit-vector with length $q-1$, representing where the splits are; a directed graph $H$ on no more than $q+m'<q(q+1)$ vertices, representing known inequalities between current keys *and* older keys retired by decrease-key operations; and finally, a mapping from elements to vertices in $H$. One may easily confirm that the state can be represented in no more than $3q^4=o(\log n)$ bits. One may also confirm that a split or decrease-key can update the state in $O(1)$ time. We now turn to the findmin operation. Consider the *findmin-action* function, which determines the next step in the findmin procedure. It can be represented as

$$\text{findmin-action} \;:\; \text{state}\times\{1,\ldots,q\} \;\rightarrow\; \big(V(H)\times V(H)\big)\cup\{1,\ldots,q\},$$

where the first $\{1,\ldots,q\}$ represents the argument to the findmin query. The findmin-action function can either perform a comparison (represented by $V(H)\times V(H)$) which, if performed, will alter the state, or return an answer to the findmin query, represented by the second $\{1,\ldots,q\}$. One simply applies the findmin-action function until it produces an answer. We will represent the findmin-action function as a table. Since the state is represented in $o(\log n)$ bits, we can keep it in one machine word; therefore, computing the findmin-action function (and updating the state) takes constant time on a RAM.

One can see that any split-findmin solver can be converted, without loss of efficiency, into one that performs comparisons only during calls to findmin. Therefore, finding the optimal findmin-action function is tantamount to finding the optimal split-findmin solver.

We have now reduced the split-findmin problem to a brute force search over the findmin-action function. There are less than $F=2^{3q^4}\cdot q\cdot(q^4+q)<2^{4q^4}$ distinct findmin-action functions, most of which do not produce correct answers. There are less than $I=(2q+q^2(q+1))^{q^2+3q}$ distinct instances of the problem, because the number of decrease-keys is $<q^2$, findmins $<2q$, and splits $<q$. Furthermore, each operation can be a split or findmin, giving the $2q$ term, or a decrease-key, which requires us to choose an element and where to fit its new key into the permutation, giving the $q^2(q+1)$ term. Each findmin-action/problem instance pair can be tested for correctness in $V=O(q^2)$ time, and therefore all correct findmin-action functions can be chosen in time $F\cdot I\cdot V=2^{\Omega(q^4)}$. For $q=\log\log n$ this is $o(n)$, meaning the time for this brute force search does not affect the other constant factors involved.

How do we choose the optimal split-findmin solver? This is actually not a trivial question because of the possibility of there not being one solver that dominates all others on all input sizes. Consider charting the worst-case complexity of a solver $\mathcal{S}$ as a function $g_{\mathcal{S}}$ of the number of operations $p$ in the input sequence. It is plausible that certain solvers are optimal for only certain densities $p/q$. We need to show that for some solver $\mathcal{S}^*$, $g_{\mathcal{S}^*}$ is within a constant factor of the lower envelope of $\{g_{\mathcal{S}}\}_{\mathcal{S}}$, where $\mathcal{S}$ ranges over all correct solvers. Let $\mathcal{S}_k$ be the optimal solver for $2^k$ operations. We let $\mathcal{S}^*$ be the solver that mimics $\mathcal{S}_k$ from operations $2^{k-1}+1$ to $2^k$. At operation $2^k$ it resets its state, reexecutes all $2^k$ operations under $\mathcal{S}_{k+1}$, and continues using $\mathcal{S}_{k+1}$ until operation $2^{k+1}$. Since $g_{\mathcal{S}_{k+1}}(2^{k+1}) \leq 2 \cdot g_{\mathcal{S}_k}(2^k)$, it follows that $g_{\mathcal{S}^*}(p) \leq 4 \cdot \min_{\mathcal{S}}\{g_{\mathcal{S}}(p)\}$.

Our overall algorithm is very simple. We divide the $n$ elements into $n' = n/q$ superelements, each representing a contiguous block of $q$ elements. Each unsplit sequence then consists of three parts: two subsequences in the leftmost and rightmost superelements and a third subsequence consisting of unsplit superelements. We use Gabow's algorithm on the unsplit superelements, where the key of a superelement is the minimum over constituent elements. For the superelements already split, we use the $\mathcal{S}^*$ split-findmin solver constructed as above. The cost of Gabow's algorithm is $O((m + n/q)\alpha(m, n/q)) = O(m + n)$, and the cost of using $\mathcal{S}^*$ on each superelement is $\Theta(\text{SPLIT-FINDMIN}(m, n))$ by construction; therefore the overall cost is $\Theta(\text{SPLIT-FINDMIN}(m, n))$.

One can easily extend the proof to randomized split-findmin solvers by defining the findmin-action as selecting a distribution over actions. $\square$

We note that the *time* bound of Theorem 4.2 on pointer machines is provably optimal. La Poutré [LaP96] gave a lower bound on the pointer machine complexity of the split-find problem, which is subsumed by the split-findmin problem. The results in this section address the RAM complexity and decision-tree complexity of split-findmin, which are unrelated to La Poutré's result.

## REFERENCES

[AGM97] N. Alon, Z. Galil, and O. Margalit, *On the exponent of the all pairs shortest path problem*, J. Comput. System Sci., 54 (1997), pp. 255–262.

[AHU76] A. V. Aho, J. E. Hopcroft, and J. D. Ullman, *On finding lowest common ancestors in trees*, SIAM J. Comput., 5 (1976), pp. 115–132.

[AMO93] R. K. Ahuja, T. L. Magnati, and J. B. Orlin, *Network Flows: Theory, Algorithms, and Applications*, Prentice–Hall, Englewood Cliffs, NJ, 1993.

[AMOT90] R. K. Ahuja, K. Mehlhorn, J. B. Orlin, and R. E. Tarjan, *Faster algorithms for the shortest path problem*, J. ACM, 37 (1990), pp. 213–223.

[BKRW98] A. L. Buchsbaum, H. Kaplan, A. Rogers, and J. R. Westbrook, *Linear-time pointer-machine algorithms for LCAs, MST verification, and dominators*, in Proceedings of the 30th ACM Symposium on Theory of Computing (STOC), Dallas, TX, 1998, ACM, New York, 1998, pp. 279–288.

[BTZ98] G. S. Brodal, J. L. Träff, and C. D. Zaroliagis, *A parallel priority queue with constant time operations*, J. Parallel and Distrib. Comput., 49 (1998), pp. 4–21.

[Chaz00] B. Chazelle, *A minimum spanning tree algorithm with inverse-Ackermann type complexity*, J. ACM, 47 (2000), pp. 1028–1047.

[CLRS01] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms*, MIT Press, Cambridge, MA, 2001.

[CMMS98] A. Crauser, K. Mehlhorn, U. Meyer, and P. Sanders, *A parallelization of Dijkstra's shortest path algorithm*, in Proceedings of the 23rd International Symposium on Mathematical Foundations of Computer Science (MFCS), Lecture Notes in Comput. Sci. 1450, Springer, New York, 1998, pp. 722–731.

[Dij59] E. W. Dijkstra, *A note on two problems in connexion with graphs*, Numer. Math., 1 (1959), pp. 269–271.

[Din78] E. A. Dinic, *Economical algorithms for finding shortest paths in a network*, Transportation Modeling Systems, (1978), pp. 36–44 (in Russian).

[Din03] Y. Dinitz, *Personal communication*, Ben-Gurion University, Be'er Sheva, Israel, 2003.

[FG85] A. M. Frieze and G. R. Grimmett, *The shortest-path problem for graphs with random arc-lengths*, Discrete Appl. Math., 10 (1985), pp. 57–77.

[FR01] J. Fakcharoenphol and S. Rao, *Planar graphs, negative weight edges, shortest paths, and near linear time*, in Proceedings of the 42nd IEEE Symposium on Foundations of Computer Science (FOCS), Las Vegas, NV, 2001, IEEE Press, Piscataway, NJ, pp. 232–241.

[F91] G. N. Frederickson, *Planar graph decomposition and all pairs shortest paths*, J. ACM, 38 (1991), pp. 162–204.

[F76] M. L. Fredman, *New bounds on the complexity of the shortest path problem*, SIAM J. Comput., 5 (1976), pp. 83–89.

[FT87] M. L. Fredman and R. E. Tarjan, *Fibonacci heaps and their uses in improved network optimization algorithms*, J. ACM, 34 (1987), pp. 596–615.

[FW93] M. L. Fredman and D. E. Willard, *Surpassing the information-theoretic bound with fusion trees*, J. Comput. System Sci., 47 (1993), pp. 424–436.

[G01] A. V. Goldberg, *A simple shortest path algorithm with linear average time*, in Proceedings of the 9th European Symposium on Algorithms (ESA), Lecture Notes in Comput. Sci. 2161, Springer, New York, 2001, pp. 230–241.

[G85a] H. N. Gabow, *A scaling algorithm for weighted matching on general graphs*, in Proceedings of the 26th IEEE Symposium on Foundations of Computer Science (FOCS), Portland, OR, 1985, IEEE Press, Piscataway, NJ, pp. 90–100.

[G85b] H. N. Gabow, *Scaling algorithms for network problems*, J. Comput. System Sci., 31 (1985), pp. 148–168.

[G95] A. V. Goldberg, *Scaling algorithms for the shortest paths problem*, SIAM J. Comput., 24 (1995), pp. 494–504.

[GM97] Z. Galil and O. Margalit, *All pairs shortest distances for graphs with small integer length edges*, Inform. and Comput., 134 (1997), pp. 103–139.

[GR98] A. V. Goldberg and S. Rao, *Beyond the flow decomposition barrier*, J. ACM, 45 (1998), pp. 783–797.

[GT89] H. N. Gabow and R. E. Tarjan, *Faster scaling algorithms for network problems*, SIAM J. Comput., 18 (1989), pp. 1013–1036.

[GT91] H. N. Gabow and R. E. Tarjan, *Faster scaling algorithms for general graph-matching problems*, J. ACM, 38 (1991), pp. 815–853.

[GYY80] R. L. Graham, A. C. Yao, and F. F. Yao, *Information bounds are weak in the shortest distance problem*, J. ACM, 27 (1980), pp. 428–444.

[Hag00] T. Hagerup, *Improved shortest paths on the word RAM*, in Proceedings of the 27th International Colloquium on Automata, Languages, and Programming (ICALP), Lecture Notes in Comput. Sci. 1853, Springer, New York, 2000, pp. 61–72.

[Hag04] T. Hagerup, *Simpler computation of single-source shortest paths in linear average time*, in Proceedings in the 21st Annual Symposium on Theoretical Aspects of Computer Science (STACS), Montpellier, France, 2004, Springer, New York, pp. 362–369.

[Han04] Y. Han, *Improved algorithm for all pairs shortest paths*, Inform. Process. Lett., 91 (2004), pp. 245–250.

[HKRS97] M. R. Henzinger, P. N. Klein, S. Rao, and S. Subramanian, *Faster shortest path algorithms for planar graphs*, J. Comput. System Sci., 55 (1997), pp. 3–23.

[HT02] Y. Han and M. Thorup, *Integer sorting in $O(n\sqrt{\log\log n})$ expected time and linear space*, in Proceedings of the 43rd Annual Symposium on Foundations of Computer Science (FOCS), Vancouver, 2002, IEEE Press, Piscataway, NJ, pp. 135–144.

[J77] D. B. Johnson, *Efficient algorithms for shortest paths in sparse networks*, J. ACM, 24 (1977), pp. 1–13.

[K70] L. R. Kerr, *The Effect of Algebraic Structure on the Computational Complexity of Matrix Multiplication*, Technical report TR70-75, Computer Science Department, Cornell University, Ithaca, NY, 1970.

[KKP93] D. R. Karger, D. Koller, and S. J. Phillips, *Finding the hidden path: Time bounds for all-pairs shortest paths*, SIAM J. Comput., 22 (1993), pp. 1199–1217.

[KKT95] D. R. Karger, P. N. Klein, and R. E. Tarjan, *A randomized linear-time algorithm for finding minimum spanning trees*, J. ACM, 42 (1995), pp. 321–329.

[KS97] P. N. Klein and S. Subramanian, *A randomized parallel algorithm for single-source shortest paths*, J. Algorithms, 25 (1997), pp. 205–220.

[KS98] S. G. Kolliopoulos and C. Stein, *Finding real-valued single-source shortest paths in $o(n^3)$ expected time*, J. Algorithms, 28 (1998), pp. 125–141.

[LaP96] H. LaPoutré, *Lower bounds for the union-find and the split-find problem on pointer machines*, J. Comput. System Sci., 52 (1996), pp. 87–99.

[M01] U. Meyer, *Single-source shortest-paths on arbitrary directed graphs in linear average-case time*, in Proceedings of the 12th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA), Washington, DC, 2001, SIAM, Philadelphia, pp. 797–806.

[M02] U. Meyer, *Buckets strike back: Improved parallel shortest-paths*, in Proceedings of the 16th International Parallel and Distributed Processing Symposium (IPDPS), Ft. Lauderdale, FL, 2002, IEEE Computer Society Press, Los Alamitos, CA, pp. 75–82.

[Mit00] J. S. B. Mitchell, *Geometric shortest paths and network optimization*, in Handbook of Computational Geometry, North–Holland, Amsterdam, 2000, pp. 633–701.

[MN00] K. Mehlhorn and S. Näher, *LEDA: A Platform for Combinatorial and Geometric Computing*, Cambridge University Press, Cambridge, UK, 2000.

[MT87] A. Moffat and T. Takaoka, *An all pairs shortest path algorithm with expected time $O(n^2 \log n)$*, SIAM J. Comput., 16 (1987), pp. 1023–1031.

[Pet02b] S. Pettie, *On the comparison-addition complexity of all-pairs shortest paths*, in Proceedings of the 13th International Symposium on Algorithms and Computation (ISAAC'02), Vancouver, 2002, Springer, New York, pp. 32–43.

[Pet03] S. Pettie, *On the Shortest Path and Minimum Spanning Tree Problems*, Ph.D. thesis, Department of Computer Sciences, The University of Texas at Austin, Austin, TX, 2003; also available online as Technical report TR-03-35 at http://www.cs.utexas.edu/ftp/pub/techreports/tr03-35.ps.gz.

[Pet04] S. Pettie, *A new approach to all-pairs shortest paths on real-weighted graphs*, Special Issue of Selected Papers from the 29th International Colloqium on Automata Languages and Programming (ICALP 2002), Theoret. Comput. Sci., 312 (2004), pp. 47–74.

[PR02a] S. Pettie and V. Ramachandran, *Computing shortest paths with comparisons and additions*, in Proceedings of the 13th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA), San Francisco, CA, 2002, SIAM, Philadelphia, pp. 267–276.

[PR02b] S. Pettie and V. Ramachandran, *Minimizing randomness in minimum spanning tree, parallel connectivity, and set maxima algorithms*, in Proceedings of the 13th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA), San Francisco, CA, 2002, SIAM, Philadelphia, pp. 713–722.

[PR02c] S. Pettie and V. Ramachandran, *An optimal minimum spanning tree algorithm*, J. ACM, 49 (2002), pp. 16–34.

[PRS02] S. Pettie, V. Ramachandran, and S. Sridhar, *Experimental evaluation of a new shortest path algorithm*, in Proceedings of the 4th Workshop on Algorithm Engineering and Experiments (ALENEX), San Francisco, CA, 2002, Springer, New York, pp. 126–142.

[Sei95] R. Seidel, *On the all-pairs-shortest-path problem in unweighted undirected graphs*, J. Comput. System Sci., 51 (1995), pp. 400–403.

[SP75] P. M. Spira and A. Pan, *On finding and updating spanning trees and shortest paths*, SIAM J. Comput., 4 (1975), pp. 375–380.

[Spi73] P. M. Spira, *A new algorithm for finding all shortest paths in a graph of positive arcs in average time $O(n^2 \log^2 n)$*, SIAM J. Comput., 2 (1973), pp. 28–32.

[SZ99] A. Shoshan and U. Zwick, *All pairs shortest paths in undirected graphs with integer weights*, in Proceedings of the 40th Annual IEEE Symposium on Foundations of Computer Science (FOCS), New York, 1999, IEEE Press, Piscataway, NJ, pp. 605–614.

[Tak92] T. Takaoka, *A new upper bound on the complexity of the all pairs shortest path problem*, Inform. Process. Lett., 43 (1992), pp. 195–199.

[Tak98] T. Takaoka, *Subcubic cost algorithms for the all pairs shortest path problem*, Algorithmica, 20 (1998), pp. 309–318.

[Tar79] R. E. Tarjan, *A class of algorithms which require nonlinear time to maintain disjoint sets*, J. Comput. System Sci., 18 (1979), pp. 110–127.

[Tar79b] R. E. Tarjan, *Applications of path compression on balanced trees*, J. ACM, 26 (1979), pp. 690–715.

[Tar82] R. E. Tarjan, *Sensitivity analysis of minimum spanning trees and shortest path trees*, Inform. Process. Lett., 14 (1982), pp. 30–33; *Corrigendum*, Inform. Process. Lett., 23 (1986), p. 219.

[Tho00] M. Thorup, *Floats, integers, and single source shortest paths*, J. Algorithms, 35 (2000), pp. 189–201.

[Tho03] M. Thorup, *Integer priority queues with decrease key in constant time and the single source shortest paths problem*, in Proceedings of the 35th Annual ACM Symposium on Theory of Computing (STOC), San Diego, CA, 2003, ACM, New York, pp. 149–158.

[Tho99] M. Thorup, *Undirected single-source shortest paths with positive integer weights in linear time*, J. ACM, 46 (1999), pp. 362–394.

[TZ96] J. L. Träff and C. D. Zaroliagis, *A simple parallel algorithm for the single-source shortest path problem on planar digraphs*, in Parallel Algorithms for Irregularly Structured Problems, Lecture Notes in Comput. Sci. 1117, Springer, New York, 1996, pp. 183–194.

[Z01] U. Zwick, *Exact and approximate distances in graphs—A survey*, in Proceedings of the 9th European Symposium on Algorithms (ESA), University of Aarhus, Denmark, 2001, pp. 33–48; available online at http://www.cs.tau.ac.il/∼zwick/.

[Z02] U. Zwick, *All pairs shortest paths using bridging sets and rectangular matrix multiplication*, J. ACM, 49 (2002), pp. 289–317.

[Z04] U. Zwick, *A slightly improved sub-cubic algorithm for the all pairs shortest paths problem with real edge lengths*, in Proceedings of the 15th International Symposium on Algorithms and Computation (ISAAC), Lecture Notes in Comput. Sci. 3341, Springer, New York, 2004, pp. 921–932.

SIAM J. COMPUT.
Vol. 34, No. 6, pp. 1432–1442

# ON THE BOUNDED SUM-OF-DIGITS DISCRETE LOGARITHM PROBLEM IN FINITE FIELDS*

QI CHENG†

**Abstract.** In this paper, we study the bounded sum-of-digits discrete logarithm problem in finite fields. Our results are concerned primarily with fields $\mathbf{F}_{q^n}$, where $n|q-1$. The fields are called Kummer extensions of $\mathbf{F}_q$. It is known that we can efficiently construct an element $g$ with order exponential in $n$. Let $S_q(\bullet)$ be the function from integers to the sum of digits in their $q$-ary expansions. We first present an algorithm that, given $g^e$ $(0 \le e < q^n)$, finds $e$ in random polynomial time, provided that $S_q(e) < n$. We then show that the problem is solvable in random polynomial time for most of the exponent $e$ with $S_q(e) < 1.32n$ by exploring an interesting connection between the discrete logarithm problem and the problem of list decoding of Reed–Solomon codes and applying the Guruswami–Sudan algorithm. As far as we are aware, our algorithm is the first one which can solve discrete logarithms of $2^{\log^{1-\epsilon} q^n}$ many instances in polynomial time for infinite many constant characteristic fields $\mathbf{F}_{q^n}$. Furthermore, since every finite field has an extension of reasonable degree, which is a Kummer extension, our result reveals an unexpected property of the discrete logarithm problem, namely, the bounded sum-of-digits discrete logarithm problem in any given finite field becomes polynomial-time solvable in certain low degree extensions.

As a side result, we obtain a sharper lower bound on the number of congruent polynomials generated by linear factors than the one based on the Stothers–Mason ABC-theorem. We also prove that, in the field $\mathbf{F}_{q^{q-1}}$, the bounded sum-of-digits discrete logarithm with respect to $g$ can be computed in random time $O(f(w)\log^4(q^{q-1}))$, where $f$ is a subexponential function and $w$ is the bound on the $q$-ary sum-of-digits of the exponent; hence the problem is fixed parameter tractable. These results are shown to be generalized to Artin–Schreier extension $\mathbf{F}_{p^p}$, where $p$ is a prime.

**Key words.** discrete logarithm problem, sum-of-digits, finite fields, parameterized complexity

**AMS subject classifications.** 11Y16, 68Q25, 68W30

**DOI.** 10.1137/S0097539704446037

## 1. Introduction and motivation.

Most practical public key cryptosystems base their security on the hardness of solving the integer factorization problem or the discrete logarithm problem in finite fields. Both problems admit subexponential algorithms; thus we have to use long parameters, which make the encryption/decryption costly if the parameters are randomly chosen. Parameters of low Hamming weight, or more generally, of small sum-of-digits, offer some remedy. Another popular approach is to use a small multiplicative subgroup of a large field [16]. Both methods speed up the system while seeming to keep the security intact. In particular, in the cryptosystem based on the discrete logarithm problem in finite fields of small characteristic, using small sum-of-digits exponents is very attractive, due to the existence of normal bases [2]. It is proposed and implemented for smart cards and mobile devices, where the computing power is severely limited. Although attacks exploring the specialty were proposed [17], none of them have polynomial-time complexity.

Let $\mathbf{F}_{q^n}$ be a finite field. For $\beta \in \mathbf{F}_{q^n}$, if $\beta, \beta^q, \beta^{q^2}, \dots, \beta^{q^{n-1}}$ form a linear basis of $\mathbf{F}_{q^n}$ over $\mathbf{F}_q$, we call them, collectively, *a normal basis.* It is known that a normal

*Received by the editors September 16, 2004; accepted for publication (in revised form) April 10, 2005; published electronically July 26, 2005. This research was partially supported by NSF Career Award CCR-0237845. A preliminary version of this paper appeared in *Advances in Cryptology: Proceedings of the 24th Annual International Cryptology Conference* (*Crypto* 2004), M. Franklin, ed., Lecture Notes in Comput. Sci. 3152, Springer, Berlin, pp. 201–212.
http://www.siam.org/journals/sicomp/34-6/44603.html

†School of Computer Science, University of Oklahoma, Norman, OK 73019 (qcheng@cs.ou. edu).

basis exists for every pair of prime power $q$ and a positive integer $n$ [12, p. 29]. Every element $\alpha$ in $\mathbf{F}_{q^n}$ can be represented as

$$\alpha = a_0\beta + a_1\beta^q + \cdots + a_{n-1}\beta^{q^{n-1}},$$

where $a_i \in \mathbf{F}_q$ for $0 \le i \le n-1$. The power of $q$ is a linear operation; thus

$$\alpha^q = a_0\beta^q + \cdots + a_{n-2}\beta^{q^{n-1}} + a_{n-1}\beta.$$

Hence to compute the $q$th power, we need only rotate the digits, which can be done very quicky, possibly on the hardware level. Let $e$ be an integer with $q$-ary expansion

$$e = e_0 + e_1 q + e_2 q^2 + \cdots + e_{n-1}q^{n-1} \qquad (0 \le e_i < q \text{ for } 0 \le i \le n-1). \tag{1.1}$$

The sum-of-digits of $e$ in the $q$-ary expansion is defined as $S_q(e) = \sum_{i=0}^{n-1} e_i$. When $q = 2$, the sum-of-digits becomes the famous Hamming weight. To compute $\alpha^e$, we need only do at most $S_q(e)$ many multiplications in addition to cyclic shiftings. Furthermore, the exponentiation algorithm can be parallelized, which is a property not enjoyed by the large characteristic fields. It was proved in [19] that the size of the arithmetic circuit computing the $e$th exponentiation is about $S_q(e)$, and the depth is about $\log_2 S_q(e)$.

The discrete logarithm problem in finite field $\mathbf{F}_{q^n}$ is to compute an integer $e$ such that $t = g^e$, given a generator $g$ of a subgroup of $\mathbf{F}_{q^n}^*$ and $t$ in the subgroup. The general purpose algorithms for solving the discrete logarithm problem are the number field sieve and the function field sieve (for a survey, see [14]). They have conjectured time complexity

$$\exp(c(\log q^n)^{1/3}(\log\log q^n)^{2/3})$$

for some constant $c$, when $q$ is small or $n$ is small.

**1.1. Related work.** Suppose we want to compute the discrete logarithm of $t = g^e$ with respect to base $g$ in the finite field $\mathbf{F}_{q^n}$. An algorithm proposed by Coppersmith (described in [17]) works well when the Hamming weight of $e$, denoted by $w$, is very small. It is a clever adaptation of the baby-step giant-step idea and runs in random time $O(\sqrt{w}\binom{\lfloor \log q^n/2\rfloor}{\lfloor w/2\rfloor})$. It is proved in [17] that the average-case complexity achieves only a constant factor speed-up over the worst case. As an interesting comparison, the fastest method for solving the subgroup discrete logarithm problem is either to solve it in the large field or to use the Pollard rho algorithm in the subgroup, which obtains a square root speed-up over the trivial exhaustive search algorithm.

It is not clear how Coppersmith's idea can be generalized when the exponent has small sum-of-digits in the base $q > 2$. However, we can consider the very special case when $e_i \in \{0, 1\}$ for $0 \le i \le n-1$ and $\sum_{0\le i\le n-1} e_i = \lfloor \frac{n}{2} \rfloor$. Recall that $e_i$'s are the digits of $e$ in the $q$-ary expansion. It can be verified that Coppersmith's algorithm can be applied in this case. The time complexity becomes $O(\sqrt{n}\binom{\lfloor n/2\rfloor}{\lfloor n/4\rfloor})$. If $q < n^{O(1)}$, it is much worse than the time complexity of the function field sieve on a general exponent.

If the $q$-ary sum-of-digits of the exponent is bounded by $w$, is there an algorithm which runs in time $f(w)\log^c(q^n)$ and solves the discrete logarithm problem in $\mathbf{F}_{q^n}$ for some function $f$ and a constant $c$? A similar problem has been proposed from the parametric point of view by Fellows and Koblitz in [11], where they consider the prime finite fields and the bounded Hamming weight exponents. Their problem is listed among the most important open problems in the theory of parameterized complexity [10]. From the above discussions, it is certainly more relevant to cryptography to treat the finite fields with small characteristic and exponents with bounded sum-of-digits.

Unlike the case of the integer factorization, where many special purpose algorithms exist, the discrete logarithm problem is considered more intractable in general. As an example, one should not use the RSA (Rivest–Shamir–Adleman) algorithm modulus of approximately 1000 bits with one prime factor of 160 bits. It would be vulnerable to the elliptic curve factorization attack. However, in the Digital Signature Standard, adopted by the U.S. Government, the finite field has cardinality of approximately $2^{1024}$ or larger, while the encryption/decryption is done in a subgroup of cardinality of approximately $2^{160}$. As another example, one should search for a secret prime as random as possible in the RSA algorithm, while in the case of the discrete logarithm problem, one may use a finite field of small characteristic; hence the group of very special order. It is believed that no trapdoor can be placed in the group order, as long as it has a large prime factor (see the panel report on this issue in the Proceeding of Eurocrypt 1992 [15]). In order to have an efficient algorithm for solving the discrete logarithm, we need every prime factor of the group order to be bounding by a polynomial function on the logarithm of the cardinality of the field. Given the current state of analytic number theory, it is very hard, if not impossible, to decide whether there exists *infinitely* many finite fields of even (or constant) characteristic, where the discrete logarithm can be solved in polynomial time.

In summary, there are several common assumptions about the discrete logarithm problem in finite fields as follows:

1. As long as the group order has a large prime factor, the discrete logarithm problem is hard. We may use exponents with small sum-of-digits, since the discrete logarithm problem in that case seems to be fixed-parameter intractable. We gain advantage in speed by using bounded sum-of-digits exponents, while at the same time we keep the problem as infeasible as when using the general exponents.
2. If computing discrete logarithm is difficult, it should be difficult for any generator of the group. The discrete logarithm problem with respect to one generator can be reduced to the discrete logarithm problem with respect to any generator. Even though a reduction is not available in the small sum-of-digits case, it is not known that changing the generator of the group affects the hardness of the discrete logarithm problem.

**1.2. Our results.** In this paper, we show that the above assumptions taken in combination are incorrect. We study the discrete logarithm problem in large multiplicative subgroups of the Kummer and Artin–Schreier extensions with a prescribed base and prove that the bounded sum-of-digits discrete logarithm is easy in those groups. More precisely, we prove constructively the following theorem.

THEOREM 1.1 (main). *There exists a random algorithm for finding the integer $e$ given $g$ and $g^e$ in $\mathbf{F}_{q^n}$ in polynomial time in $\log(q^n)$ under the following conditions:*

1. $n|q-1$;
2. $0 \le e < q^n$ *and* $S_q(e) \le n$;
3. $g = \alpha + b$, *where* $\mathbf{F}_q(\alpha) = \mathbf{F}_{q^n}$, $b \in \mathbf{F}_q^*$, *and* $\alpha^n \in \mathbf{F}_q$.

*Moreover, there does not exist an integer $e' \ne e$ satisfying that* $0 \le e' < q^n$, $S_q(e') \le n$ *and* $g^{e'} = g^e$.

The theorem leads directly to a parameterized complexity result concerning the bounded sum-of-digits discrete logarithm, which answers an important open question for many special yet nonnegligible cases.

COROLLARY 1.2. *There exists an element $g$ of order greater than $2^q$ in $\mathbf{F}^*_{q^{q-1}}$ such that the discrete logarithm problem with respect to the generator $g$ can be solved in time $f(w)\log^4(q^{q-1})$, where $f$ is a subexponential function and $w$ is the bound of the sum-of-digits of the exponent in $q$-ary expansion.*

The following few comments are in order:

- For a finite field $\mathbf{F}_{q^n}$, if $n|q-1$, then there exists $g \in \mathbf{F}_{q^n}$ satisfying the condition in the theorem; in other words, there exists an irreducible polynomial of form $x^n - a$ $(a \in \mathbf{F}_q)$ over $\mathbf{F}_q$; vice versa, if there exists $\alpha$ such that $\mathbf{F}_q(\alpha) = \mathbf{F}_{q^n}$ and $\alpha^n \in \mathbf{F}_q$, then $n|q-1$.
- As a comparison, Coppersmith's algorithm runs in exponential time in the case when $e_i \in \{0,1\}$ for $0 \le i \le n-1$, $S_q(e) = \frac{n}{2}$, and $q < n^{O(1)}$, while our algorithm runs in polynomial time in that case. On the other hand, Coppersmith's algorithm works for every finite field, while our algorithm works only in Kummer and Artin–Schreier extensions.
- Usually, we can increase the strength of the cryptosystem based on a finite field discrete logarithm problem by moving the problem to extension fields. Our result shows that we need to be careful here. As an example, suppose that to replace $\mathbf{F}_{q^n}$, where $gcd(n,q) = 1$ and $n$ does not divide $q-1$, we decide to use the field $\mathbf{F}_{q^{nl}}$, where $l$ is the order of $q$ in $(\mathbf{Z}/n\mathbf{Z})^*$. The field $\mathbf{F}_{q^{nl}} = \mathbf{F}_{(q^l)^n}$ is a Kummer extension of $\mathbf{F}_{q^l}$ since $n|q^l-1$. According to Theorem 1.1, there is a polynomial-time algorithm that computes the discrete logarithm to some element $g$ in $\mathbf{F}_{q^{ln}}$, provided that the sum-of-digits of the exponent in the $q^l$-ary expansion is less than $n$. Hence, the bounded sum-of-digits discrete logarithm problem over $\mathbf{F}_{(q^l)^n}$ is much easier than the problem over $\mathbf{F}_{q^n}$. Our result reveals the following unexpected property of the discrete logarithm problem in finite fields: the difficulty of the bounded sum-of-digits discrete logarithm problem may decrease dramatically if we move up to extensions.
- Even though we cannot bound from below the largest prime factor of the order of $g$, it seems, as supported by numerical evidence, that the order of $g$, which is a factor of $q^n - 1$ larger than $2^n$, is rarely smooth. For instance, in $\mathbf{F}_{2^{889}} = \mathbf{F}_{128^{127}}$, any $g$ generates the whole group $\mathbf{F}^*_{2^{889}}$. The order $2^{889} - 1$ contains a prime factor of 749 bits. One should not attempt to apply the Silver–Pohlig–Hellman algorithm here.

The following natural question arises: Can the restriction on the sum-of-digits in Theorem 1.1 be relaxed? Clearly, if we can solve the problem under condition $S_q(e) \le (q-1)n$ in polynomial time, then the discrete logarithm problem in the subgroup generated by $g$ is broken. If $g$ is a generator of $\mathbf{F}^*_{q^n}$, which is true in many cases, then the discrete logarithm problem in $\mathbf{F}_{q^n}$ and any of its subfields to any base are broken. We find a surprising relationship between the relaxed problem and the list decoding problem for Reed–Solomon codes. We are able to prove the following theorem.

THEOREM 1.3. *Suppose $e$ is chosen randomly from the set*

$$\{0 \le e < q^n - 1 | S_q(e) < 1.32n\}.$$

*There exists an algorithm, given $g$ and $g^e$ in $\mathbf{F}_{q^n}$, for finding $e$ in polynomial time in $\log(q^n)$, with probability greater than $1 - c^{-n}$ for some constant $c$ greater than 1, under the following conditions:*

1. $n|q-1$;
2. $g = \alpha + b$, *where* $\mathbf{F}_q(\alpha) = \mathbf{F}_{q^n}$, $b \in \mathbf{F}_q^*$, *and* $\alpha^n \in \mathbf{F}_q$.

Given a polynomial ring $\mathbf{F}_q[x]/(h(x))$, it is an important problem to determine the size of the multiplicative subgroup generated by $x - s_1, x - s_2, \ldots, x - s_n$, where $(s_1, s_2, \ldots, s_n) = S$ is a list of distinct elements in $\mathbf{F}_q$ and for all $i$, $h(s_i) \neq 0$. The lower bound of the order directly affects the time complexity of an AKS (Agrawal–Kayal–Saxena)-style primality-proving algorithm. In that context, the degree of $h(x)$ divides $n$. Assume that deg $h(x) = n$. For a list of integers $E = (e_1, e_2, \ldots, e_n)$, we denote

$$(x-s_1)^{e_1}(x-s_2)^{e_2}\cdots(x-s_n)^{e_n}$$

by $(x-S)^E$. One can estimate the number of distinct congruent polynomials of the form $(x-S)^E$ modulo $h(x)$ for $E$ in a certain set. It is obvious that if $E \in \{(e_1, e_2, \ldots, e_n) | \sum e_i < n-1, e_i \geq 0\}$, then all the polynomials are in different congruent classes. This gives a lower bound of approximately $4^n$. Through a clever use of the Stothers–Mason ABC-theorem, Voloch [18] and Bernstein [5] proved that if $\sum e_i < 1.1n$, then at most four such polynomials can fall into the same congruent class; hence a lower bound of $4.27689^n$ is obtained. We improve their result and obtain a lower bound of $5.17736^n$.

Theorem 1.4. *Use the above notation. Let $C$ be*

$$\left\{(e_1, e_2, \ldots, e_n) | e_i \geq 0 \text{ for } 1 \leq i \leq n, \sum_{i=1}^{n} e_i < 1.5501n, |\{i | e_i \neq 0\}| = \lfloor 0.7416n \rfloor\right\}.$$

*If there exist pairwise different elements $E_1, E_2, \ldots, E_m \in C$ such that*

$$(x-S)^{E_1} \equiv (x-S)^{E_2} \equiv \cdots \equiv (x-S)^{E_m} \pmod{h(x)},$$

*then $m = O(n^2)$. Note that $|C| = 5.17736^n n^{\Theta(1)}$.*

By allowing negative exponents, Voloch [18] obtained a bound of $5.828^n$. Our bound is smaller than his. However, starting from $|S| = 2\deg h(x)$, our method gives slightly better bounds [9]. A distinct feature of our bound is that it relates to the list decoding problem for Reed–Solomon codes. If a better list decoding algorithm is found, then our bound can be improved accordingly.

### 1.3. Organization of the paper.

The paper is organized as follows. In section 2, we list some results of counting numbers with small sum-of-digits. In section 3, we present the basic idea and the algorithm, and we prove Theorem 1.1 and Corollary 1.2. In section 4, we prove Theorems 1.3 and 1.4. In section 5, we extend our results to Artin–Schreier extensions. We conclude our paper with discussions of open problems.

## 2. Numbers with small sum-of-digits.

Suppose that the $q$-ary expansion of a positive integer $e$ is

$$e = e_0 + e_1 q + e_2 q^2 + \cdots + e_{n-1} q^{n-1},$$

where $0 \leq e_i \leq q-1$ for all $0 \leq i \leq n-1$. How many nonnegative integers $e$ less than $q^n$ satisfy $S_q(e) = w$? Denote the number by $N(w, n, q)$. Then $N(w, n, q)$ equals the number of nonnegative integral solutions of

$$\sum_{i=0}^{n-1} e_i = w$$

under the conditions that $0 \le e_i \le q-1$ for all $0 \le i \le n-1$. The generating function for $N(w,n,q)$ is

$$(1+x+\cdots+x^{q-1})^n = \sum_i N(i,n,q)x^i.$$

If $w \le q-1$, then the conditions $e_i \le q-1$ can be removed, and we have that $N(w,n,q) = \binom{w+n-1}{n-1}$. It is easy to see that if $q=2$, we have that $N(w,n,2) = \binom{n}{w}$. Later, we will need to estimate $N(w,n,q)$, where $w$ is $n$ times a small constant less than 2. Since

$$\begin{aligned}(1+&\ x +\cdots+x^{q-1})^n \\ &= \left(\frac{1-x^q}{1-x}\right)^n \\ &= (1-x^q)^n \sum_{i=0}^{\infty} \binom{i+n-1}{n-1} x^i \\ &\equiv (1-nx^q) \sum_{i=0}^{2q-1} \binom{i+n-1}{n-1} x^i \pmod{x^{2q}} \\ &\equiv \sum_{i=0}^{q-1} \binom{i+n-1}{n-1} x^i + \sum_{i=q}^{2q-1} \left( \binom{i+n-1}{n-1} - n\binom{i-q+n-1}{n-1} \right) x^i \pmod{x^{2q}},\end{aligned}$$

we have $N(w,n,q) = \binom{w+n-1}{n-1} - n\binom{w-q+n-1}{n-1}$ if $q \le w < 2q$.

## 3. The algorithm and the proof of the main theorem.

The basic idea of our algorithm is adopted from the index calculus algorithm. The idea of index calculus computing a discrete logarithm over finite fields was developed by Western and Miller [20] and Adleman [1]. To start the algorithm, we select a small set of elements called *the factor base*. It usually consists of prime numbers less than a chosen bound or of polynomials of low degrees. In the preprocessing part, we try to compute the discrete logarithm of elements in the factor base with respect to the base (denoted by $g$). We do this by randomly selecting an exponent $e$ and computing $g^e$, taking the result as an integer or a polynomial, and factoring it over $\mathbf{Z}$ or $\mathbf{F}[x]$. If it can be factored into elements in the factor base, i.e., if it is smooth, we obtain a linear relation between the discrete logarithms of the elements in the factor base. We repeat the step a number of times and complete the preprocessing part by solving the linear system after we accumulate enough relations. The next step is selecting random exponents $e$ and evaluating $tg^e$, where $t$ is the element whose discrete logarithm is sought. Hopefully it becomes smooth after a number of trials, in which case we obtain the discrete logarithm of $t$. For the algorithm to be efficient, we would like to have a small factor base and a high probability of a random element being smooth. These two goals contradict each other and we need to make a compromise; the resultant algorithm usually has subexponential time complexity.

Let $\mathbf{F}_{q^n}$ be a Kummer extension of $\mathbf{F}_q$, namely, $n | q-1$. Assume that $q = p^d$, where $p$ is the characteristic of the field. The field $\mathbf{F}_{q^n}$ is usually given as $\mathbf{F}_p[x]/(u(x))$, where $u(x)$ is an irreducible polynomial of degree $dn$ over $\mathbf{F}_p$. If $g$ satisfies the condition in Theorem 1.1, then $x^n - \alpha^n$ must be an irreducible polynomial over $\mathbf{F}_q$. Denote $\alpha^n$ by $a$. To implement our algorithm, it is necessary that we work in another model of $\mathbf{F}_{q^n}$, namely, $\mathbf{F}_q[x]/(x^n-a)$. Fortunately the isomorphism

$$\psi : \mathbf{F}_p[y]/(u(y)) \to \mathbf{F}_{q^n} = \mathbf{F}_q[x]/(x^n-a)$$

can be efficiently computed. To compute $\psi(v(y))$, where $v(y)$ is a polynomial of degree at most $dn-1$ over $\mathbf{F}_p$, all we have to do is factor $u(y)$ over $\mathbf{F}_q[x]/(x^n-a)$ and evaluate $v(y)$ at one of the roots. Factoring polynomials over finite fields is a well-studied problem in computational number theory; see [3] for a complete survey. The random algorithm runs in expected time $O(dn(dn+\log q^n)(dn\log q^n)^2)$, and the deterministic algorithm runs in time $O(dn(dn+q)(dn\log q^n)^2)$. From now on we assume the model $\mathbf{F}_q[x]/(x^n-a)$.

Consider the subgroup generated by $g=\alpha+b$ in $(\mathbf{F}_q[x]/(x^n-a))^*$ and recall that $b\in\mathbf{F}_q^*$ and $\alpha=x \pmod{x^n-a}$. The generator $g$ has an order greater than $4^n$ [8] and has a very nice property as follows. Denoting $a^{\frac{q-1}{n}}$ by $h$, we have

$$g^q=(\alpha+b)^q=\alpha^q+b=a^{\frac{q-1}{n}}\alpha+b=h\alpha+b;$$

more generally,

$$(\alpha+b)^{q^i}=\alpha^{q^i}+b=h^i\alpha+b.$$

In other words, we obtain a set of relations $\log_{\alpha+b}(h^i\alpha+b)=q^i$ for $0\le i\le n-1$. This corresponds to the precomputation stage of the index calculus. The difference is that, in our case, this stage finishes in polynomial time, while generally it requires subexponential time. For a general exponent $e$,

$$\begin{aligned}(\alpha+b)^e&=(\alpha+b)^{e_0+e_1q+\cdots+e_{n-1}q^{n-1}}\\&=(\alpha+b)^{e_0}(h\alpha+b)^{e_1}\cdots(h^i\alpha+b)^{e_i}\cdots(h^{n-1}\alpha+b)^{e_{n-1}}.\end{aligned}$$

If $f(\alpha)$ is an element in $\mathbf{F}_{q^n}$, where $f\in\mathbf{F}_q[x]$ is a polynomial of degree less than $n$, and $f(\alpha)=(\alpha+b)^e$ and $S_q(e)<n$, then, due to unique factorization in $\mathbf{F}_q[x]$, $f(x)$ can be completely split into linear factors over $\mathbf{F}_q$. We can read the discrete logarithm from the factorizations after the coefficients are normalized. The algorithm is described as follows.

Algorithm 1.

*Input: $g$, $g^e$ in $\mathbf{F}_{q^n}=\mathbf{F}_q[x]/(x^n-a)$ satisfying the conditions in Theorem* 1.1.

*Output: $e$.*

1. *Define an order in $\mathbf{F}_q$ (for example, use the lexicographic order). Compute and sort the list $(1,h,h^2,h^3,\ldots,h^{n-1})$.*
2. *Suppose that $g^e$ is represented by $f(\alpha)$, where $f\in\mathbf{F}_q[x]$ has degree less than $n$. Factoring $f(x)$ over $\mathbf{F}_q$, let $f(x)=c(x+d_1)^{e_1}\cdots(x+d_k)^{e_k}$, where $c,d_1,\ldots,d_k$ are in $\mathbf{F}_q$.*
3. *(Normalization.) Normalize the coefficients and reorder the factors of $f(x)$ such that their constant coefficients are $b$ and $f(x)=(x+b)^{e_1}\cdots(h_{n-1}x+b)^{e_{n-1}}$, where $h_i=h^i$.*
4. *Output $e_0+e_1q+\cdots+e_{n-1}q^{n-1}$.*

Step 1 takes time $O(n\log^2q\log n+n\log n\log q)=O(n\log n\log^2q)$. The most time-consuming part is factoring a polynomial over $\mathbf{F}_q$ with degree at most $n$. The random algorithm runs in expected time $O(n(n+\log q)(n\log q)^2)$ and the deterministic algorithm runs in time $O(n(n+q)(n\log q)^2)=O(n^3q\log^2q)$. Normalization and reordering can be done in time $O(n\log n\log q)$, since we have a sorted list of $(1,h,h^2,h^3,\ldots,h^{n-1})$. Thus, the algorithm can be finished in random time $O(n(n+\log q)(n\log q)^2)$ and in deterministic time $O(n^3q\log^2q)$. This concludes the proof of the main theorem.

Now we are ready to prove Corollary 1.2. Any $f(x)$, where $f(\alpha) = (\alpha + b)^e \in \, <\alpha + b> \subseteq \mathbf{F}_{q^{q-1}}$, is congruent to a product of at most $w = S_q(e)$ linear factors modulo $x^{q-1} - a$. If $w < q - 1$, we have an algorithm running in time $O(q^4 \log^2 q)$ and solving the discrete logarithm, according to Theorem 1.1. So we need only consider the case when $w \geq q - 1$. The general purpose algorithm will run in random time $f(\log q^{q-1})$, where $f$ is a subexponential function. Theorem 1.2 follows from the fact that $\log q^{q-1} \leq w \log w$ when $w \geq q - 1$.

There are approximately $2^{\log^{1-\epsilon} q^{q-1}}$ many nonnegative solutions of $e_0 + e_1 + \cdots + e_{q-2} < q-1$; hence our algorithm can efficiently solve discrete logarithms of $2^{\log^{1-\epsilon} q^{q-1}}$ many instances for fields $\mathbf{F}_{q^{q-1}}$.

**4. The application of the list decoding algorithm of Reed–Solomon codes.** The following natural question arises: Can we relax the bound on the sum-of-digits and still obtain a polynomial-time algorithm? Solving the problem under the condition $S_q(e) \leq (q - 1)n$ basically renders the discrete logarithm problems in $\mathbf{F}_{q^n}$ and any of its subfields easy. Suppose that $g^e = f(\alpha)$, where $f(x) \in \mathbf{F}_q[x]$ has degree less than $n$. Using the same notation as in the previous section, we have

$$f(\alpha) = (\alpha + b)^{e_0}(h\alpha + b)^{e_2} \cdots (h^{n-1}\alpha + b)^{e_{n-1}}.$$

Note that $f(x)$ may be irreducible. For example, let $\alpha = x \pmod{x^5 - 2}$ in $\mathbf{F}_{11}$,

$$(\alpha + 1)^{2+11+2*11^2+11^4} = 7\alpha^4 + 9\alpha^3 + 10\alpha^2 + 3\alpha + 9,$$

and $7x^4 + 9x^3 + 10x^2 + 3x + 9$ is irreducible over $\mathbf{F}_{11}$. In any case, there exists a polynomial $t(x)$ with degree $\sum_{i=0}^{n-1} e_i - n$ such that

$$f(x) + (x^n - a)t(x) = (x + b)^{e_0}(hx + b)^{e_1} \cdots (h^{n-1}x + b)^{e_{n-1}}.$$

We call the polynomial $t(x)$ a *smoothing polynomial* for $f(x)$. If the cardinality of $\{i | e_i \neq 0\}$ is greater than $k$, then the curve $y = t(x)$ will pass at least $k$ points in the set

$$\left\{ \left( i, -\frac{f(i)}{i^{q-1} - a} \right) \middle| i \in \left\{ -b, -\frac{b}{h}, \ldots, -\frac{b}{h^{n-1}} \right\} \right\}.$$

To find all the polynomials of degree $d = \sum_{i=0}^{n-1} e_i - n$ that pass at least $k$ points in a given set of $n$ points, we use the list decoding problem for Reed–Solomon codes. It turns out that there are only a few such polynomials, and they can be found efficiently as long as $k \geq \sqrt{nd}$.

PROPOSITION 4.1 (Guruswami–Sudan [13]). *Given $n$ distinct elements $x_0, x_1, \ldots, x_{n-1} \in \mathbf{F}_q$, $n$ values $y_0, y_1, \ldots, y_{n-1} \in \mathbf{F}_q$ and a natural number $d$, there are at most $O(\sqrt{n^3 d})$ many univariate polynomials $t(x) \in \mathbf{F}_q[x]$ of degree at most $d$ such that $y_i = t(x_i)$ for at least $\sqrt{nd}$ many points. Moreover, these polynomials can be found in random polynomial time.*

For each $t(x)$, we use the Cantor–Zassenhaus algorithm [3] to factor $f(x) + (x^n - a)t(x)$. There must exist a $t(x)$ such that the polynomial $f(x) + (x^n - a) * t(x)$ can be completely factored into a product of linear factors in $\{h^i x + b | 0 \leq i \leq n - 1\}$, and $e$ is computed as a consequence.

**4.1. The proof of Theorem 1.3.** In this section, we consider the case when $S_q(e) \leq 1.32n$. If there are at least $0.5657n \geq \sqrt{0.32n \cdot n}$ number of nonzero $e_i$'s, then we can apply the Guruswami–Sudan algorithm to find all the smoothing polynomials. In order to prove Theorem 1.3, it remains to show the following lemma.

LEMMA 4.2. *Define* $A_{n,q}$ *as*

$$\{(e_1, e_2, \ldots, e_n) \mid e_1+e_2+\cdots+e_n \leq 1.32n, e_i \in \mathbf{Z}, \text{ and } 0 \leq e_i \leq q-1 \text{ for } 1 \leq i \leq n\}$$

*and* $B_n$ *as*

$$\{(e_1, e_2, \ldots, e_n) \mid |\{i | e_i \neq 0\}| < 0.5657n\}.$$

*We have*

$$\frac{|A_{n,q} \cap B_n|}{|A_{n,q}|} < c^{-n}$$

*for some constant* $c > 1$ *when* $n$ *is sufficiently large.*

*Proof.* The cardinality of $A_{n,q}$ is $\sum_{i=0}^{\lfloor 1.32n \rfloor} N(i, n, q) > \binom{2.32n}{n} > 4.883987, \ldots^n$. The cardinality of $A_{n,q} \cap B_n$ is less than $\sum_{v=\lceil 0.5657n \rceil}^{n} \binom{n}{v}\binom{1.32n}{n-v-1}$. The summands maximize at $v = 0.5657n$ if $v \geq 0.5657n$. Hence we have

$$\begin{aligned} &\sum_{v=\lceil 0.5657n \rceil}^{n} \binom{n}{v}\binom{\lfloor 1.32n \rfloor}{n-v-1} \\ &\qquad < 0.4343n \binom{n}{\lceil 0.5657n \rceil}\binom{\lfloor 1.32n \rfloor}{\lfloor 0.4343n \rfloor} \\ &\qquad < 4.883799, \ldots^n. \end{aligned}$$

This proves the lemma with $c = 4.883987, \ldots / 4.883799, \ldots > 1$. ◻

**4.2. The proof of Theorem 1.4.**

*Proof.* Let $\tau$ be a positive real number less than 1. Define

$$C_{n,q,\tau} = \left\{ (e_1, e_2, \ldots, e_n) \left| \begin{array}{l} e_1 + e_2 + \cdots + e_n = \lfloor (1+\tau)n \rfloor, e_i \in \mathbf{Z} \\ \text{and } 0 \leq e_i \leq q-1 \text{ for } 1 \leq i \leq n \\ \text{and } |\{i | e_i \neq 0\}| = \lfloor \sqrt{\tau} n \rfloor \end{array} \right. \right\}.$$

Given $f(x) \in \mathbf{F}_q[x]$, if there exists $E \in C_{n,q,\tau}$, such that $(x-S)^E \equiv f(x) \pmod{h(x)}$, there must exist a polynomial $t(x)$ such that $(x-S)^E = t(x)h(x) + f(x)$, and $t(x)$ is a solution for the list decoding problem with input $\{(s, -\frac{f(s)}{h(s)}) | s \in S\}$. According to Proposition 4.1, there are at most $O(n^2)$ solutions. Thus the number of congruent classes modulo $h(x)$ that $\{(x-S)^E | E \in C_{n,q,\tau}\}$ has is $\Omega(|C_{n,q,\tau}|/n^2)$. We have

$$|C_{n,q,\tau}| = \binom{n}{\sqrt{\tau} n}\binom{(1+\tau)n}{\sqrt{\tau} n} = n^{\Theta(1)} \left( \frac{(1+\tau)^{1+\tau}}{\tau^{\sqrt{\tau}}(1-\sqrt{\tau})^{1-\sqrt{\tau}}(1+\tau-\sqrt{\tau})^{1+\tau-\sqrt{\tau}}} \right)^n.$$

It takes the maximum value $n^{\Theta(1)} 5.17736, \ldots^n$ at $\tau = 0.5501$. ◻

**5. Artin–Schreier extensions.** Let $p$ be a prime. The Artin–Schreier extension of a finite field $\mathbf{F}_p$ is $\mathbf{F}_{p^p}$. The polynomial $x^p - x - a = 0$ is irreducible in $\mathbf{F}_p$

for any $a \in \mathbf{F}_p^*$, since $\alpha, \alpha^p, \ldots, \alpha^{p^{p-1}}$ are all different and $\alpha^{p^p} = \alpha$, where $\alpha = x \pmod{x^p - x - a}$. So we may take $\mathbf{F}_{p^p} = \mathbf{F}_p[x]/(x^p - x - a)$. For any $b \in \mathbf{F}_p$, we have

$$(\alpha + b)^p = \alpha^p + b = \alpha + b + a,$$

and similarly

$$(\alpha + b)^{p^i} = \alpha^{p^i} + b = \alpha + b + ia.$$

Hence the results for Kummer extensions can be adopted to Artin–Schreier extensions. For the subgroup generated by $\alpha + b$, we have a polynomial algorithm for solving the discrete logarithm if the exponent has $p$-ary sum-of-digits less than $p$. Note that $b$ may be 0 in this case.

THEOREM 5.1. *There exists an algorithm for finding the integer $e$ given $g$ and $g^e$ in $\mathbf{F}_{p^p}$ in polynomial time in $\log p^p$ under the following conditions:*

1. $0 \le e < p^p$ *and* $S_q(e) \le p - 1$;
2. $g = \alpha + b$, *where* $\mathbf{F}_p(\alpha) = \mathbf{F}_{p^p}$, $b \in \mathbf{F}_p$, *and* $\alpha^p + \alpha \in \mathbf{F}_p^*$.

*Moreover, there does not exist an integer $e' \neq e$ satisfying that $0 \le e' < p^p$, $S_q(e') \le n$ and $g^{e'} = g^e$.*

THEOREM 5.2. *There exists an element $g$ of order greater than $2^p$ in $\mathbf{F}_{p^p}^*$ such that the discrete logarithm problem with respect to $g$ can be solved in time $O(f(w)(\log p^p)^4)$, where $f$ is a subexponential function and $w$ is the bound of the sum-of-digits of the exponent in the $p$-ary expansion.*

THEOREM 5.3. *Suppose that $g = \alpha + b$, where $\mathbf{F}_p(\alpha) = \mathbf{F}_{p^p}$, $b \in \mathbf{F}_p$, and $\alpha^p + \alpha \in \mathbf{F}_p^*$. Suppose $e$ is chosen in random from the set*

$$\{0 \le e < q^n - 1 | S_q(e) < 1.32n\}.$$

*There exists an algorithm given $g$ and $g^e$ in $\mathbf{F}_{p^p}$ for finding $e$ in polynomial time in $\log(p^p)$, with probability greater than $1 - c^{-n}$ for some constant $c$ greater than* 1.

## 6. Concluding remarks.

A novel idea in the celebrated AKS primality testing algorithm is to construct a subgroup of large cardinality through linear elements in finite fields. The subsequent improvements [6, 7, 4] rely on constructing a single element of large order. It is speculated that these ideas will be useful in attacking the integer factorization problem. In this paper, we show that they do affect the discrete logarithm problem in finite fields. We give an efficient algorithm that computes the bounded sum-of-digits discrete logarithm with respect to prescribed bases in Kummer extensions. We also show that the problem is related to an important problem in coding theory. The most interesting open problem is to further relax the restriction on the sum-of-digits of the exponent.

**Acknowledgment.** We thank Professor Pedro Berrizbeitia for very helpful discussions.

## REFERENCES

[1] L. M. ADLEMAN, *A subexponential algorithm for the discrete logarithm problem with applications to cryptography*, in Proceedings of the 20th IEEE Symposium on Foundations of Computer Science, 1979, pp. 55–60.

[2] G. B. AGNEW, R. C. MULLIN, I. M. ONYSZCHUK, AND S. A. VANSTONE, *An implementation for a fast public-key cryptosystem*, J. Cryptology, 3 (1991), pp. 63–79.

[3] E. Bach and J. Shallit, *Algorithmic Number Theory*, vol. I, The MIT Press, Cambridge, MA, 1996.

[4] D. J. Bernstein, *Proving Primality in Essentially Quartic Random Time*, http://cr.yp.to/papers.html#quartic, 2003. Math. Comp., to appear.

[5] D. J. Bernstein, *Sharper ABC-based Bounds for Congruent Polynomials*, http://cr.yp.to/papers.html#abccong, 2003. Journal de Theor. Nombres Bordeaux, to appear.

[6] P. Berrizbeitia, *Sharpening "Primes Is in P" for a Large Family of Numbers*, http://lanl.arxiv.org/abs/math.NT/0211334, 2002. Math. Comp., to appear.

[7] Q. Cheng, *Primality proving via one round in ECPP and one iteration in AKS*, in Proceedings of the 23rd Annual International Cryptology Conference (CRYPTO), Lecture Notes in Comput. Sci. 2729, D. Boneh, ed., Springer-Verlag, Berlin, 2003, pp. 338–348.

[8] Q. Cheng, *Constructing finite field extensions with large order elements*, in Proceedings of the 15th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA), SIAM, Philadelphia, 2004, pp. 1123–1124.

[9] Q. Cheng, K. Vuchi, and Y.-H. Li, *On the Size of the Subgroup Generated by Linear Factors*, manuscript, 2004.

[10] R. G. Downey and M. R. Fellows, *Parameterized Complexity*, Springer-Verlag, New York, 1999.

[11] M. Fellows and N. Koblitz, *Fixed-parameter complexity and cryptography*, in Proceedings of the Tenth International Symposium on Applied Algebra, Algebraic Algorithms and Error-Correcting Codes (AAECC '93), Lecture Notes in Comput. Sci. 673, Springer-Verlag, Berlin, 1993, pp. 121–131.

[12] S. Gao, *Normal Bases over Finite Fields*, Ph.D. thesis, The University of Waterloo, Ontario, Canada, 1993.

[13] V. Guruswami and M. Sudan, *Improved decoding of Reed–Solomon and algebraic-geometry codes*, IEEE Trans. Inform. Theory, 45 (1999), pp. 1757–1767.

[14] A. M. Odlyzko, *Discrete logarithms: The past and the future*, Des. Codes Cryptogr., 19 (2000), pp. 129–145.

[15] R. A. Rueppel, *Advances in Cryptology—EUROCRYPT* '92*, Proceedings of the Workshop on Theory and Application of Cryptographic Techniques,* Lecture Notes in Comput. Sci. 658, Springer-Verlag, Berlin, 1993.

[16] C. P. Schnorr, *Efficient signature generation by smart cards*, J. Cryptology, 4 (1991), pp. 161–174.

[17] D. R. Stinson, *Some baby-step giant-step algorithms for the low Hamming weight discrete logarithm problem*, Math. Comp., 71 (2002), pp. 379–391.

[18] J. F. Voloch, *On some subgroups of the multiplicative group of finite rings*, J. Théor. Nombres Bordeaux, 16 (2004), pp. 233–239.

[19] J. von zur Gathen, *Efficient exponentiation in finite fields*, in Proceedings of the 32nd Annual IEEE Symposium on Foundations of Computer Science, San Juan, PR, 1991, pp. 384–391.

[20] A. E. Western and J. C. P. Miller, *Tables of Indices and Primitive Roots*, Vol. 9, Cambridge University Press, London, 1968.

SIAM J. COMPUT.
Vol. 34, No. 6, pp. 1443–1463

# DUALITY BETWEEN PREFETCHING AND QUEUED WRITING WITH PARALLEL DISKS[*]

DAVID A. HUTCHINSON[†], PETER SANDERS[‡], AND JEFFREY SCOTT VITTER[§]

**Abstract.** Parallel disks promise to be a cost effective means for achieving high bandwidth in applications involving massive data sets, but algorithms for parallel disks can be difficult to devise. To combat this problem, we define a useful and natural duality between writing to parallel disks and the seemingly more difficult problem of prefetching. We first explore this duality for applications involving read-once accesses using parallel disks. We get a simple linear time algorithm for computing optimal prefetch schedules and analyze the efficiency of the resulting schedules for randomly placed data and for arbitrary interleaved accesses to striped sequences. Duality also provides an optimal schedule for prefetching plus caching, where blocks can be accessed multiple times. Another application of this duality gives us the first parallel disk sorting algorithms that are provably optimal up to lower-order terms. One of these algorithms is a simple and practical variant of multiway mergesort, addressing a question that had been open for some time.

**Key words.** caching, external memory sorting, load balancing, lower bound, prefetching, randomized algorithm

**AMS subject classifications.** 68W10, 68W20, 68W40, 68M20, 68P10, 68P20, 68Q17

**DOI.** 10.1137/S0097539703431573

## 1. Introduction.

External memory (EM) algorithms are those for which the problem data set is too large to fit into the high-speed random access memory (RAM) of a computer and therefore must reside on external devices such as disk drives [23]. In order to cope with the high latency of accessing data on disks, efficient EM algorithms exploit locality in their design. In the *I/O model*, EM algorithms access a large *block* of $B$ contiguous data elements in one *I/O step* and perform the necessary algorithmic operations on the elements in the block while in the high-speed memory. The speedup can be significant. However, even with blocked access, a single disk provides much less bandwidth than the internal memory. This problem can be mitigated by using multiple disks in parallel. For each input/output operation, one block is transferred between a fast memory of size $M$ and each of the $D$ disks. The algorithm therefore transfers $D$ blocks at the cost of a single-disk access delay.

A simple approach to algorithm design for parallel disks is to employ large logical blocks, or *superblocks*, of size $B \cdot D$ in the algorithm. This reduces the problem to designing an EM algorithm for one disk with logical block size $BD$. A superblock is split into $D$ physical blocks—one on each disk. All $D$ physical blocks are accessed

[*]Received by the editors July 18, 2003; accepted for publication (in revised form) March 21, 2005; published electronically August 17, 2005. A substantial part of this work was done while the authors were at Duke University.
http://www.siam.org/journals/sicomp/34-6/43157.html

[†]Department of Systems and Computer Engineering, Carleton University, Ottawa K1S 5B6, ON, Canada (hutchins@sce.carleton.ca). This author's research was supported in part by the National Science Foundation through research grant CCR–0082986.

[‡]Fakultät für Informatik, Universität Karlsruhe, 76128 Karlsruhe, Germany (sanders@ira.uka.de). This author's research was partially supported by the Future and Emerging Technologies programme of the EU under contract IST-1999-14186 (ALCOM-FT).

[§]Department of Computer Sciences, Purdue University, West Lafayette, IN 47907–2066 (jsv@purdue.edu). This author's research was supported in part by the NSF through research grants CCR–9877133 and EIA–9870724 and by the Army Research Foundation through MURI grant DAAH04–96–1–0013 and grants DAAD19–01–1–0725 and DAAD19–03–1–0321.

simultaneously whenever the superblock is accessed. We refer to this technique as *superblock striping*. Unfortunately, this approach is suboptimal for em algorithms like sorting that deal with many blocks at the same time. For sorting and many related EM problems, an optimal algorithm requires *independent access* to the $D$ disks, in which each of the $D$ blocks in a parallel I/O operation can reside at a different position on its disk [25, 23]. Designing algorithms for independent parallel disks has been surprisingly difficult [25, 21, 20, 10, 11, 5, 6, 23, 22, 24]. In this paper we consider parallel disk output and parallel disk input separately, in particular as the parallel *output scheduling problem* and the parallel *prefetch scheduling problem*, respectively.

The (online) *output scheduling (or queued writing) problem* takes as input a fixed size pool of $m$ (initially empty) memory buffers each capable of storing a block, and the sequence $\langle w_0, w_1, \ldots, w_{L-1}\rangle$ of $L$ block *write requests* as they are issued. Each write request is prelabeled with the disk it will use. The solution of the output scheduling problem is a schedule that specifies when the blocks are output (i.e., the contents of each parallel output operation). The buffer pool can be used to reorder the outputs with respect to the logical writing order given by $\langle w_0, w_1, \ldots, w_{L-1}\rangle$ so that the total number of parallel output steps is minimized.

We use the term *write* for the logical process of moving a block from the responsibility of the application to the responsibility of the scheduling algorithm. The scheduling algorithm orchestrates the physical *output* of these blocks to disks.

The (offline) *prefetch scheduling problem* takes as input a fixed size pool of $m$ (empty) memory buffers for storing blocks, and the sequence $\langle r_0, r_1, \ldots, r_{L-1}\rangle$ of $L$ distinct block *read requests* that will be issued. Each read request is prelabeled with the disk it will use. The resulting *prefetch schedule* specifies when the blocks should be fetched so that they can be consumed by the application in the right order.

By the term *read*, we mean the logical process of moving a block from the responsibility of the scheduling algorithm to the application. We use the term *fetch* (or *prefetch*) to refer to the physical disk access.

The central theme in this paper is the *duality* between these two problems. Roughly speaking, an output schedule corresponds to a prefetch schedule with reversed time axis, and vice versa. The power of this idea is that computations in one domain can be analyzed via duality with respect to computations in the other domain.

In section 2, we formally introduce the duality principle for the case of distinct blocks to be written or read (*write-once* and *read-once* scheduling). In section 3, we derive an optimal write-once output scheduling algorithm and apply the duality principle to obtain an optimal read-once prefetch scheduling algorithm. In section 4, we modify the previous algorithm so that blocks are fetched as early as possible, so as to be more robust against delays in practical implementations.

For difficult input sequences, an optimal schedule might use parallel disks very inefficiently because most disks might still be idle most of the time. In section 5, we therefore give performance guarantees for two particular classes of input sequences: randomly placed data and arbitrarily interleaved data streams. A data stream is a sequence of blocks that is read or written sequentially by the application. Many algorithms access several such streams in an interleaved manner, and the order of accesses to the streams is not predictable at the time the streams are allocated. Nevertheless, we obtain performance guarantees for the following allocation strategies of the data streams:

**Fully randomized (FR):** Each block is allocated to a random disk.

**Striping (S):** Consecutive blocks of a data stream are allocated to consecutive disks

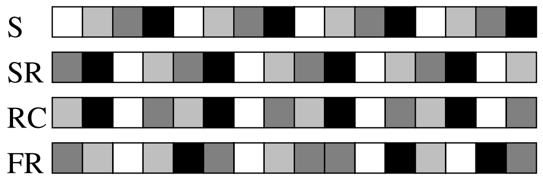


FIG. 1.1. *A sequence of* 16 *blocks allocated to* 4 *disks (*1 = *white,* 2 = *light grey,* 3 = *dark grey,* 4 = *black) using different allocation strategies.*

in a simple, round-robin manner.

**Simple randomized (SR):** For each data stream, this strategy follows a striping allocation, where the disk selected for the first block is chosen randomly and independently of the other data streams.

**Randomized cycling (RC):** Each data stream $i$ chooses a random (and independent) permutation $\pi_i$ of disk numbers and allocates the $j$th block of stream $i$ on disk $\pi_i(j \bmod D)$.

Figure 1.1 gives an example.

In section 6, we relax the restriction that blocks are accessed only once and allow caching of blocks and repeated block requests (*write-many* and *read-many* scheduling). Again we derive a simple optimal algorithm for the writing case and obtain an optimal algorithm for the reading case using the duality principle. A similar result has been obtained by Kallahalla and Varman [16, 17] using more complicated arguments.

In section 7, we apply the results from sections 3 and 5 to parallel disk sorting. Results on online writing translate into improved sorting algorithms using the distribution paradigm. Results on offline reading translate into improved sorting algorithms based on multiway merging. By appending a "D" for distribution sort or an "M" for mergesort to an allocation strategy (FR, S, SR, RC) we obtain a descriptor for a sorting algorithm (FRD, FRM, SD, SM, SRD, SRM, RCD, RCM). This notation is an extension of the notation used in [24]. RCD and RCM turn out to be particularly efficient. Let

$$\mathrm{Sort}(N) = \frac{N}{DB}\left(1 + \log_{M/B}\frac{N}{M}\right).$$

In section 8, we show that $2 \cdot \mathrm{Sort}(N)$ is the lower bound for sorting $N$ elements on $D$ disks. Our versions of RCD and RCM are the first algorithms that provably match this bound up to a lower-order term if $M = \omega(DB)$. The good performance of RCM is particularly interesting. The question of whether there is a simple variant of mergesort that is asymptotically optimal for multiple disks has been open since the model was formalized in [25]. A summary of the notation used in this paper is included in the appendix.

**Related work.** An announcement [14] and a preliminary version [13] of this paper have appeared in conference volumes. The problems of prefetching and caching have been intensively studied and can be quite difficult. We begin our overview with offline algorithms for the I/O model. Belady [7] solved the caching problem for a single disk. Kallahalla and Varman [15, 17] developed an optimal parallel disk prefetching algorithm for read-once sequences. Besides a simpler algorithm and analysis, our contribution is a linear time algorithm (which is, however, also implicit in [16], which was published simultaneously with the first announcement of our result [14].) Moreover, the concept of duality allows us to translate performance guarantees for writing into performance guarantees for reading. The main contribution in [16] is an optimal result for prefetching plus caching. Again, the concept of duality yields a simpler algorithm and proof.

Cao et al. [9] and Kimbrel and Karlin [18] have introduced a model that allows us to study integrated prefetching and caching with overlapping of I/O and computation. In this *penalty model*, internal computation is linked to I/Os by a penalty of $F$ time units for an I/O step. For $F \rightarrow \infty$, the penalty model becomes equivalent to the I/O model since the internal computations become insignificant. Kimbrel and Karlin [18] already introduced the idea of time reversal and the *reverse aggressive* algorithm that has our algorithm as a special case. They also defined a similar kind of duality, namely between fetches and evictions of a caching algorithm. The analysis in the penalty model predicts that the performance ratio between reverse aggressive and the optimal algorithm goes to infinity as $F \rightarrow \infty$. Hence it is a bit surprising that the algorithm turns out to be *optimal* in the I/O model.

The prudent prefetching algorithm introduced in section 4 is similar to the *conservative* algorithm described in [9]. The main difference is that it applies to the optimal parallel disk prefetching algorithm rather than to the optimal schedule in a sequential system.

Albers, Garg, and Leonardi [4] gave an optimal polynomial time offline algorithm for the single-disk case in the penalty model, but it does not generalize well to multiple disks. Albers and Büttner [3] overcame this problem by requiring synchronized parallel disk access (as in the I/O model) and by postulating $\mathcal{O}(D)$ additional buffer blocks not available to the optimal algorithm. Both these algorithms are based on linear programming and hence are quite complicated and time consuming.

There has also been intensive work on online integrated prefetching and caching. Albers [2] showed for a single disk that a lookahead for the next $\Omega(M/B)$ *different* blocks is needed to get good competitiveness. For parallel disks, Kallahalla and Varman [15, 17] showed that another factor of $\Omega(D)$ lookahead is needed even for the read-once problem. Applying the optimal offline algorithm to the lookahead matches this lower bound for read-once sequences.

There are deterministic algorithms for parallel disk external sorting [21, 20] that are optimal up to a constant factor, but the constant factors are not ideal. The first optimal (up to a constant factor independent of the parameters $N$, $M$, $B$, and $D$) sorting algorithm was a randomized one by Vitter and Shriver [25]. Barve, Grove, and Vitter [5] and Barve and Vitter [6] introduced a simple and efficient randomized sorting algorithm called *simple randomized mergesort (SRM)*. For each run, SRM allocates blocks to disks using the SR allocation discipline. For $\gamma < 1$, SRM comes within an additive term of about $\gamma\text{Sort}(N)$ of the sorting lower bound if $M/B = \Omega(D\log(D)/\gamma^2)$, but for $M/B = o(D\log D)$, the bound proven is not asymptotically optimal. It is an open problem whether SRM or another variant of striped mergesort could be asymptotically optimal for small internal memory. Knuth [19, Exercise 5.4.9–31] gives the question of a tight analysis of SR a difficulty rating of 48 on a scale between 1 and 50.

Sanders, Egner, and Korst [22] analyzed a (slightly) suboptimal output scheduling algorithm for FR allocation that would yield a good parallel disk distribution sorting algorithm (FRD) yet has the disadvantage that reading cannot be done in a striped fashion. To overcome the apparent difficulty of analyzing SR, Vitter and Hutchinson [24] analyzed RC allocation, which provides more randomness but retains the advantages of striping. RCD is an asymptotically optimal distribution sort algorithm that allocates successive blocks of a bucket to the disks according to the RC discipline. The present paper uses the concept of duality to apply these results to external mergesort.

The lower bound in section 8 is a refinement of the analysis by Aggarwal and Vitter [1]. In particular, our analysis gives the precise constant factor in the leading term, and it applies to algorithms that do not necessarily use the same number of inputs and outputs. The remaining gap between the upper and lower bound is only a lower-order term if $M = \omega(DB)$.

Building on the results in the present paper, Dementiev and Sanders [12] develop a parallel disk external sorting algorithm that perfectly overlaps I/O and computation. This algorithms works independently of the failure penalty $F$ and need not know how much internal work is done between I/O requests. (These times are far from constant and not easy to predict so that previous results on prefetching in the penalty model are inapplicable for sorting.)

**2. The duality principle.** Duality is a quite simple yet powerful concept once the model is properly defined. Therefore, we start with a more formal description of the model.

Our machine model is the (independent parallel disk) I/O model of Vitter and Shriver [25] with a single[1] processor, $D$ disks, and an internal memory of size $M$. All blocks have the same size $B$. In one *I/O step*, one block on each disk can be accessed in a synchronized fashion. We consider either a queued writing or a buffered prefetching arrangement, where a pool of $m$ block buffers is available to the algorithm (see Figure 2.1).

DEFINITION 2.1. *A* write-once output scheduling problem *is defined by a sequence $\Sigma = \langle b_0, \ldots, b_{L-1} \rangle$ of distinct blocks which are to be output using parallel output operations. Let* $\text{disk}(b_i)$ *denote the disk on which block $b_i$ is to be located. An application* writes *these blocks in the order specified by $\Sigma$. We use the term* write *for the logical process of moving a block from the responsibility of the application to the responsibility of the scheduling algorithm. The scheduling algorithm orchestrates the physical* output *of these blocks to disks. Time is measured in I/O steps actually performed. In particular, in each time step at least one block is output.*

*An output schedule is specified by giving a function* $\text{oStep} : \{b_0, \ldots, b_{L-1}\} \to \mathbb{N}$ *that specifies for each disk block $b_i \in \Sigma$ the time step when it will be output. An output schedule is* correct *if the following conditions hold:*

(i) *No disk is referenced more than once in a single time step. That is, if $i \neq j$ and* $\text{disk}(b_i) = \text{disk}(b_j)$, *then* $\text{oStep}(b_i) \neq \text{oStep}(b_j)$.

(ii) *The buffer pool is large enough that it does not overflow. That is, if we define*

$$\text{o}\mathcal{B}\text{acklog}(b_i) = |\{j < i : \text{oStep}(b_j) \geq \text{oStep}(b_i)\}|$$

*to be the number of blocks $b_j$ that are written before block $b_i$ but not output before $b_i$, then we require for all $0 \leq i \leq L$ that* $\text{o}\mathcal{B}\text{acklog}(b_i) < m$.

*The blocks are output in increasing order of* oStep. *The number of steps needed by an output schedule is $T = \max_{0 \leq i < L} \text{oStep}(b_i)$. An output schedule is* optimal *if it minimizes $T$ among all correct schedules.*

[1]Our results generalize to multiple processors as long as data exchange between processors is much faster than disk access.

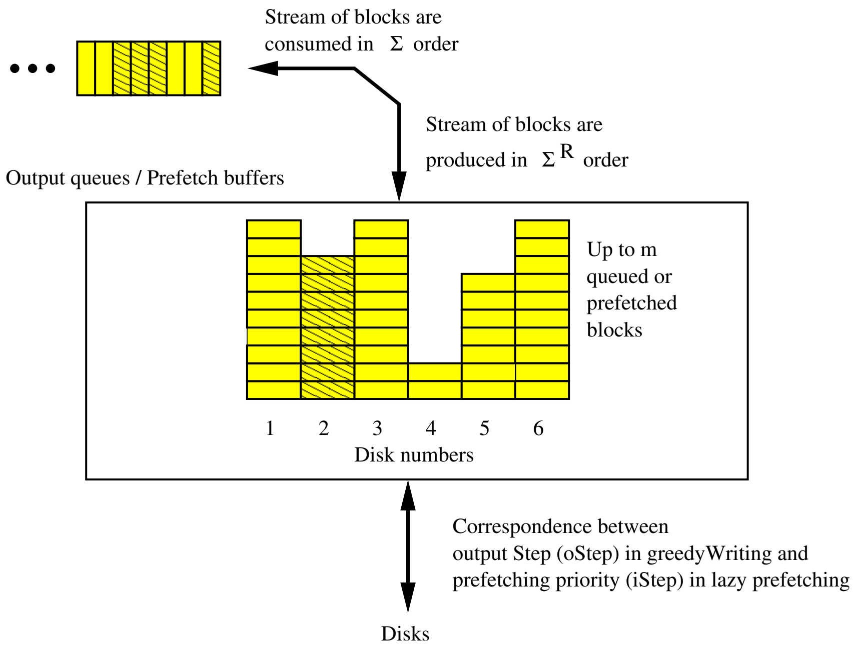

Stream of blocks are consumed in Σ order
Stream of blocks are produced in Σ R order
Output queues / Prefetch buffers
Up to m queued or prefetched blocks
1 2 3 4 5 6
Disk numbers
Correspondence between output Step (oStep) in greedyWriting and prefetching priority (iStep) in lazy prefetching
Disks

FIG. 2.1. *Duality between the prefetching priority and the output step. The hashed blocks illustrate how the blocks of disk* 2 *might be distributed.*

It will turn out that our write-once output scheduling algorithms work optimally even if they are given the blocks *online*, that is, one at a time without specifying $\Sigma$ explicitly.

DEFINITION 2.2. *Analogously to a write-once output scheduling problem, a* read-once prefetch scheduling problem *is defined by a sequence* $\Sigma$ *of blocks to be read. By the term* reading, *we mean the logical process of moving a block from the responsibility of the scheduling algorithm to the application. We use the term* fetching *(or* prefetching*) to refer to the physical disk access.*

*A prefetch schedule is defined using a function* $\text{iStep} : \{b_0, \ldots, b_{L-1}\} \to \mathbb{N}$. *The blocks are prefetched in increasing order of* iStep. *Let us define*

$$\text{i}\mathcal{B}\text{acklog}(b_i) = |\{j > i : \text{iStep}(b_j) \leq \text{iStep}(b_i)\}|$$

*to be the number of blocks* $b_j$ *that are fetched no later than block* $b_i$ *but are read after* $b_i$. *All blocks in* $\text{i}\mathcal{B}\text{acklog}(b_i)$ *must be buffered. The limited buffer pool size requires the correctness condition* $\text{i}\mathcal{B}\text{acklog}(b_i) < m$. *The number of steps needed by a prefetch schedule is* $T = \max_{0 \leq i < L} \text{iStep}(b_i)$. *A prefetch schedule is* optimal *if it minimizes* $T$ *among all correct schedules.*

It will turn out that our prefetch scheduling algorithms work *offline*; that is, they need to know $\Sigma$ in advance. We explain in section 7 how this is sufficient for sorting applications.

The following theorem shows that reading and writing not only have similar models but are equivalent to each other in a quite interesting sense.

THEOREM 2.3 (duality principle). *Consider any sequence* $\Sigma = \langle b_0, \ldots, b_{L-1} \rangle$ *of distinct write requests. Let* oStep *denote a correct output schedule for* $\Sigma$ *that uses* $T$ *output steps. Then we get a correct prefetch schedule* iStep *for* $\Sigma^R = \langle b_{L-1}, \ldots, b_0 \rangle$

*that uses $T$ fetch steps by setting* $\mathrm{iStep}(b_i) = T - \mathrm{oStep}(b_i) + 1$.

*Vice versa, every correct prefetch schedule* iStep *for $\Sigma^R$ that uses $T$ fetch steps yields a correct output schedule* $\mathrm{oStep}(b_i) = T - \mathrm{iStep}(b_i) + 1$ *for $\Sigma$, using $T$ output steps.*

*Proof.* For the first part, consider the schedule $\mathrm{iStep}(b_i) = T - \mathrm{oStep}(b_i) + 1$. The resulting fetch steps are between 1 and $T$ and all blocks on the same disk get different fetch steps. It remains to be shown that $\mathrm{i}\mathcal{B}\mathrm{acklog}(b_i) < m$ for $0 \leq i < L$. With respect to $\Sigma^R$, we have

$$\begin{aligned}\mathrm{i}\mathcal{B}\mathrm{acklog}(b_i) &= |\{j > i : \mathrm{iStep}(b_j) \leq \mathrm{iStep}(b_i)\}| \\ &= |\{j > i : T - \mathrm{oStep}(b_j) + 1 \leq T - \mathrm{oStep}(b_i) + 1\}| \\ &= |\{j > i : \mathrm{oStep}(b_j) \geq \mathrm{oStep}(b_i)\}|\,.\end{aligned}$$

The latter value is $\mathrm{o}\mathcal{B}\mathrm{acklog}(b_i)$ with respect to $\Sigma$. It is smaller than $m$ because oStep is a correct schedule.

The proof for the converse case is completely analogous. ∎

## 3. Optimal write-once and read-once scheduling.

We now give an optimal algorithm for writing a write-once sequence, prove its optimality, and then apply the duality principle to transform it into a read-once prefetching algorithm.

Consider the algorithm *greedyWriting* for writing a sequence $\Sigma = \langle b_0, \ldots, b_{L-1}\rangle$ of distinct blocks. Let $Q$ denote the set of blocks in the buffer pool so, initially, $Q = \emptyset$. Let $Q_d = \{b \in Q : \mathrm{disk}(b) = d\}$ denote the blocks queued for disk $d$. Write the blocks $b_i$ in sequence as follows:

1. If $|Q| < m$, then simply insert $b_i$ into $Q$.
2. Otherwise, each disk $d$ with $Q_d \neq \emptyset$ outputs the block of $Q_d$ that appears first in $\Sigma$. The blocks so output are then removed from $Q$ and $b_i$ is inserted into $Q$.
3. Once all blocks are written, the queues are flushed; that is, additional output steps are performed until $Q$ is empty.

Figure 3.1 gives an example.

A schedule is called a *FIFO schedule* if blocks are output in arrival order on each disk. For the write-once case, the following lemma tells us that when we look for optimal schedules, it is sufficient to consider FIFO schedules. On real disks, FIFO schedules are not necessarily optimal, since they do not optimize seek times and rotational delays, but in our synchronous model, using FIFO suffices and simplifies subsequent proofs.

LEMMA 3.1. *For any sequence of blocks $\Sigma$ and every correct output schedule* oStep *there is a FIFO output schedule* oStep$'$ *consisting of at most the same number of output steps.*

*Proof.* The proof of the lemma is based on transforming a non-FIFO schedule into a FIFO schedule by exchanging blocks in the schedule of a disk that are output out of order. Consider a non-FIFO schedule that services two block requests $b_i$ and $b_j$ for the same disk "out of order"; that is, we have $i < j$ but $\mathrm{oStep}(b_i) > \mathrm{oStep}(b_j)$. If we swap the output order of $b_i$ and $b_j$, then the buffer pool consumption between output steps $\mathrm{oStep}(b_j)$ and $\mathrm{oStep}(b_i)$ can only decrease or remain the same. Such swapping operations can be repeated as necessary until a FIFO schedule is obtained. ∎

Algorithm greedyWriting is one way to compute a FIFO schedule. The following lemma shows that greedyWriting outputs every block as early as possible.

LEMMA 3.2. *For any sequence of blocks* $\Sigma$ *and any FIFO output schedule* $\mathrm{oStep}'$ *for* $|\Sigma|$*, let* oStep *denote the schedule produced by algorithm greedyWriting. Then for all* $b_i \in \Sigma$*, we have* $\mathrm{oStep}(b_i) \le \mathrm{oStep}'(b_i)$.

*Proof.* The proof is by induction on $|\Sigma|$.

**Base case for induction.** $|\Sigma| = 0$. The claim is vacuously true for $|\Sigma| = 0$.

**Induction hypothesis.** Assume that the claim is true for $|\Sigma| = L - 1$.

**Induction step** $|\Sigma| = L-1 \rightsquigarrow |\Sigma| = L$. Consider the state of *greedyWriting* when the last block $b_L$ is scheduled. Let $d = \mathrm{disk}(b_L)$. Using the induction hypothesis, it suffices to prove that $\mathrm{oStep}(b_L) \le \mathrm{oStep}'(b_L)$.

*Case* 1. $\mathrm{oStep}(b_L) = 1$. This case is trivial since 1 is the smallest possible output step.

*Case* 2. $\mathrm{oStep}(b_L) = \mathrm{oStep}(b_l) + 1$ for some other block $b_l$ with $\mathrm{disk}(b_l) = d$. Applying the induction hypothesis to $|\Sigma| = \langle b_1, \ldots, b_{L-1}\rangle$, we have $\mathrm{oStep}(b_l) \le \mathrm{oStep}'(b_l)$. By the definition of FIFO output schedules, we have $\mathrm{oStep}'(b_l) < \mathrm{oStep}'(b_L)$; that is, $\mathrm{oStep}'(b_l) + 1 \le \mathrm{oStep}'(b_L)$. All in all, we get

$$\mathrm{oStep}(b_L) = \mathrm{oStep}(b_l) + 1 \le \mathrm{oStep}'(b_l) + 1 \le \mathrm{oStep}'(b_L).$$

**All remaining cases.** Let $t = \mathrm{oStep}(b_L) > 1$. Disk $d$ is idle during step $t-1$, and we have to explain why this is unavoidable. Let $C = \{b_l : \mathrm{oStep}(b_l) \ge t-1\}$ denote the set of blocks that are queued during step $t-1$. We must have $|C| \ge m$, since otherwise greedyWriting would have queued and output $b_L$ already during step $t-1$ or earlier. Assume that $\mathrm{oStep}(b_L) > \mathrm{oStep}'(b_L)$; that is, $\mathrm{oStep}(b_L) - 1 \ge \mathrm{oStep}'(b_L)$. By the induction hypothesis, $\mathrm{oStep}'(b_l) \ge \mathrm{oStep}(b_l)$ for all $b_l \in C$. In other words,

$$\mathrm{oStep}'(b_l) \ge \mathrm{oStep}(b_l) \ge \mathrm{oStep}(b_L) - 1 \ge \mathrm{oStep}'(b_L).$$

Hence, if the schedule defined by $\mathrm{oStep}'$ is used, all blocks in $C$ are written no earlier than $b_L$, which requires that more than $m$ blocks have to be buffered at the same time, contradicting the assumption that $\mathrm{oStep}'$ is correct. □

Combining Lemmas 3.1 and 3.2 we see that greedyWriting gives us optimal schedules for write-once sequences.

THEOREM 3.3. *Algorithm greedyWriting gives a correct, minimum length output schedule for any write-once reference sequence* $\Sigma$.

Combining the duality principle and the optimality of greedyWriting, we get an optimal algorithm for read-once prefetching that we call *lazy prefetching.*

COROLLARY 3.4. *An optimal prefetch schedule* iStep *for a sequence* $\Sigma$ *can be obtained by using greedyWriting to get an output schedule* oStep *for* $\Sigma^R$ *and setting* $\mathrm{iStep}(b_i) = T - \mathrm{oStep}(b_i) + 1$.

The schedule can be computed in time $\mathcal{O}(L + D)$ using very simple data structures. Figure 3.1 (top) gives an example. We refer to this approach as *lazy prefetching.*

## 4. Prudent prefetching.

Although the lazy prefetching approach in the previous section allows us to obtain a prefetch schedule with a minimal number of steps by means of reversing time, it has the practical disadvantage that blocks are accessed as late as possible even if most blocks could be fetched earlier. For example, in Figure 3.1 (top) only the bottom-most disk fetches a block in Step 1. This policy may result in unnecessary delays in real implementations where the access times to the blocks fluctuate. Many of these delays might be avoidable if some blocks were fetched earlier. One might instead use "eager prefetching" [6, 15], i.e., always accessing the

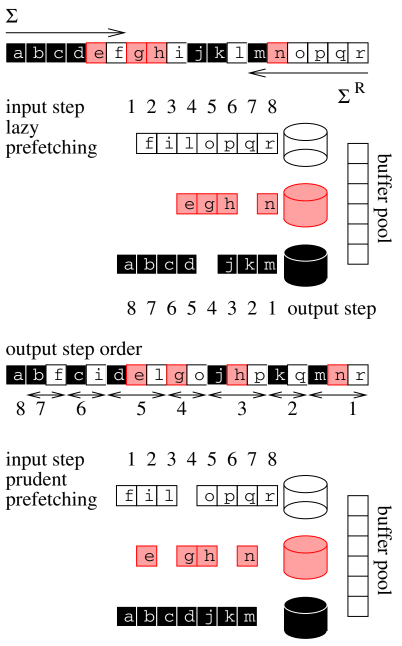


FIG. 3.1. *Duality between prefetching and output for a sequence* $\Sigma = \langle \texttt{a}, \texttt{b}, \ldots, \texttt{r} \rangle$ *of* $L = 18$ *blocks to be read using* $D = 3$ *disks and* $m = 6$ *buffers. Top part: A lazy prefetch schedule for* $\Sigma$ *as the dual of an output schedule for* $\Sigma^R$*. The shading of the blocks indicates the disks where the blocks are located. Before output Step* 2*, the eight blocks* `hijklopq` *would have to be buffered in order to output block* `h` *in Step* 2*. But since only six blocks can be buffered, the middle disk has to remain idle in Step* 2*. Similarly, before output Step* 4*, the seven blocks* `defgilo` *would have to be buffered to output block* `d`*. Bottom part: The resulting schedule for prudent prefetching. For example, before Step* 3*, blocks* `a` *and* `b` *are fetched and buffers for blocks* `fcidel` *are reserved. Block* `g` *cannot be fetched because no buffer is reserved for it. Prudent prefetching using the reading order as a priority instead of the priorities based on the optimal lazy schedule would need one more I/O step.*

highest priority block on each disk. But eager prefetching sometimes has to discard and refetch blocks, causing complications and inefficiencies.

Here we propose *prudent prefetching*, a prefetching strategy that avoids both problems. It maintains optimal schedule length, but attempts to fetch blocks as early as possible. The idea is to use the oStep obtained by greedyWriting as a priority rather than as a direct indication of the input step for fetching a block. Algorithm prudent prefetching allows blocks to be fetched before blocks with higher priority are fetched but only if buffers have been reserved for them. This way, otherwise idle disks can prefetch low priority blocks without hindering any fetches of higher priority blocks in later steps.

This strategy is easy to implement. Prudent prefetching works with a sequence $\langle l_0, \ldots, l_{L-1} \rangle$ of block requests sorted by nonincreasing priority (and hence by nondecreasing iStep of lazy prefetching). Blocks $l_0, \ldots, l_{j-1}$ have either been fetched or have a reserved buffer while blocks $l_j, \ldots, l_{L-1}$ are neither fetched nor have a reserved buffer.

Before each fetch step, all empty buffers are reserved for the next blocks in the priority sequence, and $j$ is advanced by the number of buffers so reserved. The highest priority block from each disk is then fetched if a buffer has been reserved for it. Then from each disk the highest priority block is fetched if a buffer has been reserved for it.

As before, blocks are delivered to the application in the order prescribed by $\Sigma$. When a block is delivered to the application, its buffer becomes empty and is available.

An example of the algorithm is shown in Figure 3.1. We cannot expect algorithm *prudent prefetching* to be better than lazy prefetching as long as we only count I/O steps, but we can show that it is not worse.

THEOREM 4.1. *For any correct output schedule* oStep*,* prudent prefetching *takes no more I/O steps than* lazy prefetching*.*

*Proof.* We have already observed that fetching a reserved block can never hinder a higher priority block from being fetched. Hence, in the $i$th step, all unfetched blocks with iStep $i$ will be fetched. We omit a trivial, more detailed proof by induction over the number of steps. $\square$

Another advantage of *prudent prefetching* is that it can be implemented in an event driven manner, and the fetch steps for each disk need not be synchronized. When the next block from $\Sigma$ is delivered to the application, its buffer can immediately be used for advancing $j$. When a disk finishes fetching a block, it waits (if necessary) until the next highest priority block on this disk has a reserved buffer and then starts to fetch this block. Thus there is never a need to synchronize the disks, the system can adapt to variances in access times, and the load of the interconnections between disks and processors is better balanced than for synchronous access.

## 5. How good is optimal?

When we have complex data access patterns, the knowledge that we have an optimal prefetching algorithm is often of little help. We also want to know "how good is optimal?". In the worst case, all requests may go to the same disk and no prefetching algorithm can cure the dreadful performance caused by this bottleneck. However, the situation is different if an appropriate block allocation strategy is used; for example, if blocks are allocated to disks using striping, randomization,[2] or both.

THEOREM 5.1. *Consider a sequence of $L$ block requests, and a buffer pool of size $m \geq D$ blocks. The number of I/O steps needed by greedyWriting or lazy prefetching is given by the following bounds, depending on the block allocation strategy. For striping and randomized cycling, an arbitrary interleaving of sequential accesses to $S$ sequences is allowed:*

$$
\begin{aligned}
&\textit{S:}\left\lfloor \frac{L}{D} \right\rfloor + S \quad \textit{if } m > S(D-1);\\
&\textit{FR:}\left(1+\mathcal{O}\left(\frac{D}{m}\right)\right)\frac{L}{D}+\mathcal{O}\left(\frac{m}{D}\log m\right) \qquad \textit{(expected)};\\
&\textit{RC:}\left(1+\mathcal{O}\left(\frac{D}{m}\right)\right)\frac{L}{D}+\min\left\{S+\frac{L}{D},\mathcal{O}\left(\frac{m}{D}\log m\right)\right\} \textit{ (expected).}
\end{aligned}
$$

*For the case of writing, the second term can be dropped if we are only interested in the number of steps needed to write (but not necessarily output) all blocks.*

*Proof.* Due to our result on duality, it suffices to prove the bounds for writing.

**Striping (S).** Since greedyWriting is optimal, it suffices to analyze the following specialized algorithm: Each sequence gets an exclusive allotment of $D-1$ buffer blocks. When a block from sequence $k$ is written there are two possible cases. If the pool for $k$ has a free buffer block, the block is buffered there. Otherwise, we have

[2] In practice, this will be done using simple hash functions. However, for the analysis we assume that we have a perfect source of randomness.

exactly $D$ consecutive blocks from the striped sequence $k$ so that we can output one block from sequence $k$ to each disk. There can be at most $\lfloor L/D \rfloor$ of these output steps. When all blocks are written, one additional output step for each sequence suffices to empty all buffers.

**Fully random allocation (FR).** Since *greedyWriting* is optimal, it dominates the algorithm analyzed in [22]. This algorithm admits $(1-\epsilon)D$ blocks into the buffer pool before each output step. It is shown that with this regime the buffer pool size remains in $\mathcal{O}(D/\epsilon)$ most of the time. More precisely, the probability that the required pool size exceeds $qD$ is less than $e^{(\ln 2-\epsilon q)D}$ [22, Lemma 3]. By setting $\epsilon = \Theta(D/m)$ we can make sure that the pool size is exceeded so rarely that we could afford to flush the queues whenever this happens. We omit the straightforward calculations with the tail bound showing that after an expected number of $(1+\mathcal{O}(D/m))L/D$ output steps all blocks have been written. The number of output steps needed to flush the buffers at the end is the maximum number of blocks queued at a disk. In [22] it is shown that the probability that the queue length at a particular disk exceeds $q$ is bounded by $2e^{-\epsilon q}$. Setting $q = \ln(2Dm)/\epsilon$ and multiplying by $D$, we can see that the probability $p_{\text{fail}}$ that some disk has a final load of more than $\ln(2Dm)/\epsilon$ is bounded by $D \cdot 2e^{-\epsilon \ln(2Dm)/\epsilon} = 1/m$. Since the load cannot exceed $L$, the expected maximum load is bounded by $\ln(2Dm)/\epsilon + L/m = \mathcal{O}\big(\frac{m}{D}\log m + L/m\big)$. The term $L/m$ can be absorbed into $\big(1+\mathcal{O}\big(\frac{D}{m}\big)\big)\frac{L}{D}$.

**Randomized cycling (RC).** Vitter and Hutchinson [24] show that the algorithm from [22] performs at least as well on RC-streams as it does for fully random allocation. This extends to *greedyWriting* by Theorem 3.3. Furthermore, the reverse of a sequence accessing RC-streams is indistinguishable from a sequence accessing RC-streams in the forward direction. Theorem 2.3 therefore extends the result to prefetching. Hence, the bound

$$\left(1+\mathcal{O}\left(\frac{D}{m}\right)\right)\frac{L}{D}+\mathcal{O}\left(\frac{m}{D}\log m\right)$$

transfers from the fully random allocation case.

For small $L$ and $m$ this bound can be improved using the observation that the maximum number of blocks queued at a disk at the end cannot exceed the total number of blocks allocated to it. The bound for striping shows that this load cannot exceed $L/D + S$. ◻

## 6. Prefetching with caching.

We now relax the condition that the read requests in $\Sigma$ are for distinct blocks, permitting the possibility of saving disk accesses by keeping previously accessed blocks in memory. For this *read-many* problem, we get a tradeoff for the use of the buffer pool because it has to serve the double purposes of keeping blocks that are accessed multiple times, and decoupling physical and logical accesses to equalize transient load imbalance of the disks. We define the *write-many* problem in such a way that the duality principle from Theorem 2.3 transfers: *The latest instance of each block must be kept either on its assigned disk, or in the buffer pool. The final instance of each block must be output to its assigned disk.*[3]

We prove that the following offline algorithm *manyWriting* minimizes the number of output operations for the write-many problem: Let $Q$ denote the set of blocks in

[3]The requirement that the latest versions have to be kept might seem odd in an offline setting. However, this makes sense if there is a possibility that there are reads at unknown times that need an up-to-date version of a block.

the buffer pool, so initially $Q = \emptyset$. Let $Q_d = \{b \in Q : \text{disk}(b) = d\}$ denote the blocks queued for disk $d$. To write block $b_i$, if $b_i \in Q$, the old version is overwritten in its existing buffer. Otherwise, if $|Q| < m$, $b_i$ is inserted into $Q$. If this also fails, an output step is performed before $b_i$ is inserted into $Q$. The output analogue of Belady's MIN rule [7] is used on each disk; that is, each disk with $Q_d \neq \emptyset$ outputs the block in $Q_d$ that is written again farthest in the future.

THEOREM 6.1. *Algorithm manyWriting solves the* write-many problem *with the fewest number of output steps.*

Applying duality, we also get an optimal algorithm for prefetching plus caching of a sequence $\Sigma$; using the same construction as in Corollary 3.4 we get an optimal prefetching and caching schedule.

COROLLARY 6.2. *The dual of manyWriting solves the* read-many problem *with the fewest number of input steps.*

It remains to prove Theorem 6.1.

*Proof of Theorem* 6.1. We generalize the proof of Belady's algorithm by Borodin and El-Yaniv [8] to the case of writing and multiple disks. Let $\Sigma = \langle b_0, \ldots, b_{L-1} \rangle$ be any sequence of blocks to be written. The proof is based on the following claim.

*Claim.* Let ALG be any algorithm for the write-many problem. Let $d$ denote a fixed disk. For any $0 \leq i < L$ it is possible to construct an offline algorithm $\text{ALG}_i$ that satisfies the following properties:

(i) $\text{ALG}_i$ processes the first $i - 1$ writes exactly as ALG does.

(ii) If block $b_i$ is the first block written after output step $s$, then immediately before output step $s$ there was no free buffer slot.

(iii) If $b_i$ is the first block written after output step $s$, then $\text{ALG}_i$ performs this output according to the MIN rule on disk $d$.

(iv) $\text{ALG}_i$ takes no more steps than ALG.

Once this claim is established, the theorem can be proven as follows: Starting with an optimal offline algorithm OPT, we apply the claim with $i = 0$ and $d = 0$ to obtain another optimal algorithm $\text{OPT}_0$, then apply the claim with $i = 1$ and $d = 0$ to obtain $\text{OPT}_1$ and so on. By induction over $i$, it can be seen that $\text{OPT}_{L-1}$ never leaves unused buffer slots before an output step and uses MIN for deciding which blocks to output on disk 0. Subsequently, we apply this sequence of $L$ transformations for each disk. Since these transformations do not undo property (iii) for other disks, we arrive at an optimal algorithm that works like manyWriting on all disks.

It remains to prove the claim. We initialize $\text{ALG}_i$ to ALG and transform $\text{ALG}_i$ until it fulfills all four properties. Note that this initialization automatically fulfills properties (i) and (iv). If properties (ii) and (iii) also hold, we are done.

If property (ii) is violated by $\text{ALG}_i$, then $b_i$ is the first block written by $\text{ALG}_i$ *after* some output step $s$, and before output step $s$ a free buffer slot was available. In this case, $\text{ALG}_i$ is modified so that $b_i$ is now the last block written *before* output step $s$. Note that this transformation preserves the order in which blocks are written and properties (i) and (iv). This transformation is repeated until $\text{ALG}_i$ also satisfies property (ii).

If properties (i), (ii), and (iv) hold but property (iii) is violated, there must be a write step $s$ so that $b_{i-1}$ is the last block written before output step $s$, and $b_i$ is the first block written after output step $s$ in $\text{ALG}_i$. Now we define a modified algorithm $\text{ALG}'_i$ that mimics $\text{ALG}_i$ (and hence ALG) until $b_{i-1}$ is written but uses the MIN rule in step $s$ so that properties (i)–(iii) hold for $\text{ALG}'_i$.

It remains to define the behavior of $\text{ALG}'_i$ after step $s$ so that property (iv) is also maintained. We use $\mathcal{X} + b$ as a shorthand for $\mathcal{X} + \{b\}$ for a set of blocks $\mathcal{X}$ and a block $b$. Immediately after output step $s$, the buffer pool of $\text{ALG}_i$ can be written as $\mathcal{M} = \mathcal{X} + b$ whereas $\text{ALG}'_i$ has buffer pool $\mathcal{M}' = \mathcal{X} + b'$ where $b$ is the block on disk $d$ whose next access is farthest in the future. $\text{ALG}'_i$ mimics $\text{ALG}_i$ as far as possible; that is, it performs output steps at the same time as $\text{ALG}_i$ and outputs the same blocks. As long as neither $b$ nor $b'$ is written or output by $\text{ALG}_i$, these two blocks remain the only difference between $\mathcal{M}$ and $\mathcal{M}'$. There are only three types of events that require special treatment.

**Event 1.** $\text{ALG}_i$ outputs $b$. In that case, $\text{ALG}'_i$ outputs $b'$. Afterwards we have $\mathcal{M} = \mathcal{M}'$, and from now on $\text{ALG}'_i$ can completely mimic $\text{ALG}_i$.

**Event 2.** Block $b'$ is rewritten. By definition of $b$, this situation happens before block $b$ is rewritten. After $b'$ is rewritten, we have $\mathcal{M} = \mathcal{Y} + b + b'$ and $\mathcal{M}' = \mathcal{Y} + b'$ for some common set $\mathcal{Y}$ of blocks. In particular, $\mathcal{M}'$ has one unused buffer slot. If $\text{ALG}_i$ outputs $b$ in this situation, $\text{ALG}'_i$ can again unify the states $\mathcal{M}$ and $\mathcal{M}'$ by *not* outputting anything on disk $d$.

**Event 3.** Block $b$ is rewritten. As discussed above, if $\mathcal{M} \neq \mathcal{M}'$, we must have $\mathcal{M} = \mathcal{Y} + b + b'$ and $\mathcal{M}' = \mathcal{Y} + b'$ before $b$ is rewritten. Now $\text{ALG}'_i$ uses its free buffer slot to accommodate $b$. We get $\mathcal{M} = \mathcal{M}' = \mathcal{Y} + b + b'$.

We end up with an algorithm $\text{ALG}'_i$ that fulfills properties (i)–(iv) and hence set $\text{ALG}_i{=}\text{ALG}'_i$. $\square$

**7. Application to sorting.** In this section we extend the duality between prefetching and queued writing to apply to problems of merging and distribution. In the merging phase of mergesort, there are several sorted runs on the disk, and the problem is to merge them together into a single sorted run. We assume that each run is striped across the disks using any given striping discipline, such as RC or FR, as described in the introduction. How to lay out the runs so as to permit fully parallel I/O is a challenging problem; recent work is surveyed in [23].

A big problem is that the input order $\Sigma$ for the blocks, namely the order in which the blocks need to be accessed, is highly data-dependent. The key to duality is to characterize $\Sigma$ in a simple and easily implementable way. If we examine the process of merging, as illustrated in Figure 7.1 from the bottom to top, we see that the merging buffer contains a partially filled block from each run (not yet expired). When the block empties all its items into the merged output stream, the next block from that run is inserted into the merging buffer. The merging buffer is pictured in the upper rectangle in Figure 7.1, which is distinct from the space reserved for the prefetch buffers (lower rectangle in Figure 7.1).

The first moment, therefore, that a block absolutely must be in memory is when the smallest key value of the block is merged into the output stream. We therefore define the *trigger* of a block to be its smallest key value. We say that a block is read when it is moved from the prefetch buffer to the merging buffer, where it stays until its items are exhausted by the merging process. Thus, the read order $\Sigma$ of the blocks is given by the sorted order of the triggers.

We have now reduced the merging problem to that of prefetching for the input sequence $\Sigma$. The dual problem to merging is distribution. To solve it via the duality principle, we need to process $\Sigma$ in reverse order. We equate the notion of bucket in distribution with that of run-in merging, so each block therefore has a bucket assigned to it. Since each bucket uses a fixed striping discipline, the blocks can then be assigned to disks. The dual distribution problem is thus well defined, and we get an optimal algorithm for merging by applying the algorithm of section 3.

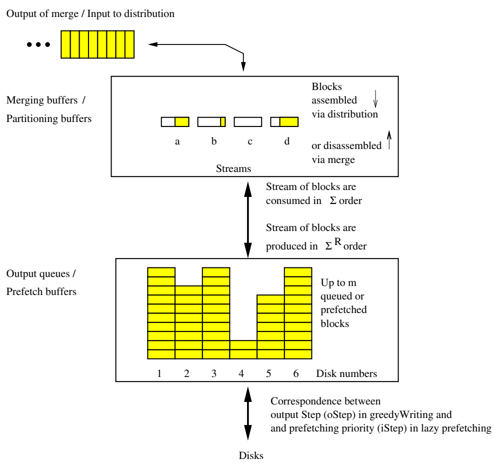

Output of merge / Input to distribution
Merging buffers / Partitioning buffers
Blocks assembled via distribution
or disassembled via merge
a
b
c
d
Streams
Stream of blocks are consumed in Σ order
Stream of blocks are produced in Σ^R order
Output queues / Prefetch buffers
Up to m queued or prefetched blocks
1 2 3 4 5 6 Disk numbers
Correspondence between output Step (oStep) in greedyWriting and and prefetching priority (iStep) in lazy prefetching
Disks

FIG. 7.1. *The relationship between merging and distribution. Buffer space is required both "privately" within the application (for storing the lead block of each run in merging, and for storing the next block being formed for each bucket in distribution), and for the Output queues / Prefetch buffers required for the techniques proposed in this paper. During distribution, the priorities of blocks correspond to their output step. For merging, blocks are read in the order given by the triggers. When an appropriate allocation discipline is used to allocate blocks of a stream to the parallel disks, the queued I/O techniques of this paper permit I/O complexity results for distribution sort to be applied to mergesort (and vice versa if desired).*

**Mergesort with randomized cycling (RCM).** How the blocks of each run are striped depends on the particular allocation discipline used. We start by discussing multiway mergesort using randomized cycling allocation (RCM) in some detail and then survey a number of additional results. Originally, the $N$ input elements are stored as a single data stream using any kind of striping. During *run formation* the input is read in chunks of size $M$ that are sorted internally and then written out in runs allocated using RC allocation. Neglecting trivial rounding issues, run formation is easy to do using $2N/(DB)$ I/O steps. For example, we need $\mathcal{O}\big(N/(DB^2)\big)$ additional I/Os for writing the trigger values to separate files. Then we set aside a buffer pool of size $m = D/\gamma$ for some parameter $\gamma$ and perform $\lceil \log_{M/B-\mathcal{O}(D/\gamma)} \frac{N}{M} \rceil$ *merge phases.* In a merge phase, groups of $k = \frac{M}{B} - \mathcal{O}(D/\gamma)$ runs are merged into new sorted runs; that is, after the last merge phase, only one sorted run is left. Merging $k$ runs of total size $sB$ can be performed using $s$ block reads by keeping one block of each run in the internal memory of the sorting application. The I/O schedule for a merging phase is found by sorting the triggers for groups of $k$ runs each. These sorted trigger sequences are then concatenated, yielding the order in which the blocks are to be read. At this point we can apply duality.

The overhead for this precomputation of $\Sigma$ (trigger values) is $\mathcal{O}\big(N/B^2\big)$ I/Os even for a single disk [6]. The triggers allow us to do optimal prefetching so that Theorem 5.1 gives an upper bound of

$$(1+\mathcal{O}(\gamma))\,\frac{N}{BD} + \min\left\{k + \frac{N}{BD}, \mathcal{O}(\log(D/\gamma)/\gamma)\right\}$$

for the expected number of fetch steps of a phase. The number of output steps for a phase is $N/(BD)$ if we have an additional output buffer of $D$ blocks. The final result

TABLE 7.1

*Summary of the I/O complexity for parallel disk sorting algorithms. Each algorithm's I/O complexity is given by* $\mathrm{Sort}_{\Delta}^{a,f}(N)$ *when the parameters are set according to the algorithm's entry in the table. Algorithms with* **boldface** *names are asymptotically optimal: M = Mergesort. SM/SD = Merge / Distribution sort with any S allocation. SRM and SRD use SR. RCD, RCD+, and RCM use RC allocation.*

| $\mathrm{Sort}_{\Delta}^{a,f}(N)$ I/Os | | | Algorithm | Source |
|---|---|---|---|---|
| $a$ | $f$ | $\Delta$ | | |
| 2 | 0 | 0 | Lower bound | |
| Deterministic algorithms | | | | |
| 2 | 0 | 0 | M, $D = 1$ | [1] |
| $\mathcal{O}(1)$ | 0 | 0 | **Greed sort** | [21] |
| $2+\gamma$ | 0 | $\Theta\left((2D)^{1+\frac{2}{\gamma}}\right)$ | M, superblock striping | |
| $2+\gamma$ | 0 | $\Theta\left((2D)^{1+\frac{2}{\gamma}}\right)$ | SM | here |
| $2+\gamma$ | 0 | $\Theta\left((2D)^{1+\frac{2}{\gamma}}\right)$ | SD | here |
| Randomized algorithms | | | | |
| $2+\gamma$ | 0 | $\Theta(D\log(D)/\gamma^2)$ | SRM | [6] |
| $2+\gamma$ | 0 | $\Theta(D\log(D)/\gamma^2)$ | SRD | here |
| $3+\gamma$ | 0 | $\Theta(D/\gamma)$ | **RCD** | [24] |
| $2+\gamma$ | $\min(\frac{N}{BD}, \log(D)/\mathcal{O}(\gamma))$ | $\Theta(D/\gamma)$ | **RCM** | here |
| $2+\gamma$ | 0 | $\Theta(D/\gamma)$ | **RCD+** | here |

is written using any striped allocation strategy; the application calling the sorting routine need not be able to handle RC allocation. For any constant $\gamma > 0$, we can write the resulting total number of I/O steps as $\mathrm{Sort}_{m+D}^{2+\mathcal{O}(\gamma),\min(\frac{N}{BD},\frac{\log D}{\mathcal{O}(\gamma)})}(N)$, where

$$\mathrm{Sort}_{\Delta}^{a,f}(N) = \frac{2N}{DB} + a \cdot \frac{N}{DB} \cdot \left\lceil \log_{\frac{M}{B}-\Delta} \frac{N}{M} \right\rceil + f + o\left(\frac{N}{DB}\right).$$

Table 7.1 compares a selection of sorting algorithms using this generalized form of the I/O bound for parallel disk sorting. The term $\frac{2N}{DB}$ represents the reading and writing of the input and the final output, respectively. The factor $a$ is decisive for the I/O complexity for large inputs. Note that for $a = 2$ and $f = \Delta = 0$ this expression is the lower bound for sorting. The additive offset $f$ may dominate for small inputs. The reduction of the memory by $\Delta$ blocks in the base of the logarithm is due to memory that is used for output or prefetching buffer pools outside the merging or distribution routines, and hence reduces the number of data streams that can be handled concurrently. One way to interpret $\Delta$ is to view it as the amount of *additional* memory needed to match the performance of the algorithm on the multihead I/O model [1] where load balancing disk accesses is not an issue.

Note that the RCM algorithm outlined above is the first asymptotically optimal parallel disk sorting algorithm that approaches the optimal constant factor 2 for $M/B \gg D$. The first two rows of Table 7.1 show that single disk sorting (e.g., multiway mergesort) is optimal. Greed sort [21] is an optimal (up to a constant factor) deterministic sorting method based on mergesort; it does it an approximate merge and then finalizes the merge using columnsort. Balance sort [20] is an equally optimal but more practical deterministic sorting algorithm that uses distribution sorting together with adaptive allocation of blocks. An algorithm frequently used in practice is a single disk algorithm together with superblock striping (i.e., using logical blocks of size $BD$). This algorithm is quite good if the input is sufficiently small that we can still sort in two passes despite the much larger block size. Using Theorem 5.1, we get the same asymptotic bounds if we use the parallel disk mergesort outlined above together with any deterministic striping discipline (SM); that is, even as a deterministic algorithm, our algorithm has performance comparable to algorithms used in practice.

Mergesort using SRM was analyzed in [6]. Although our optimal prefetching result simplifies and improves the prefetching algorithms given there, we do not get improved asymptotic bounds.

**Distribution sort with randomized cycling (RCD+).** Using random sampling and the duality between reading and writing, as shown in Figure 7.1, we can transfer the results for mergesort to results using distribution-based sorting algorithms. We obtain a new distribution sort using deterministic striping (RD) or simple randomized striping (SRD). We can also improve the analysis of the variant with RC [24] reducing the constant factor from three to two. Furthermore, an additional optimization can be used to remove the additive term $\min(\frac{N}{BD}, \log(D)/\mathcal{O}(\gamma))$ in the complexity of RCM. Below we describe the resulting algorithm RCD+ in more detail since it currently represents the parallel disk sorting algorithm with the best known bounds. The same algorithm underlies the results for the other allocation strategies SD and SRD.

The basic idea behind distribution sort is to use a generalization of quicksort where elements are classified into $k = \mathcal{O}(M/B)$ classes based on $k-1$ splitter elements. The splitters are chosen in such a way that each class has size $\mathcal{O}(N/k)$. These classes are then sorted recursively and the results are appended to form the final output.

As in mergesort, we start with an input that is striped over the disks using some arbitrary allocation strategy. We set $k = \min(N/(M - cBD), M/B - cBD)$ for an appropriate parameter $c$. To find the splitter elements we take $dk - 1$ random sample elements for an appropriate integer $d$. The sample is sorted and every $d$th element in the sorted sample is used as a splitter. Standard calculations using Chernoff bounds indicate that $d = \mathcal{O}(\log k)$ is sufficient to ensure that with high probability at most $\mathcal{O}(N/k)$ elements lie between two splitters. It can be seen that the number of I/Os needed for obtaining the sample is only a lower-order term compared to the number of I/Os needed to scan the input.

Now the input is classified into $k$ classes by scanning the input and putting each element in the appropriate class. For each class we use an output stream allocated using RC. For each class, an output buffer block is maintained that is written to an RC allocated output stream when the buffer is completely filled. Writing uses greedyWriting. Here it is useful that the algorithm is an online algorithm since it is not known in advance in what order blocks have to be written out.

The additive term in the I/O bound for RCM mergesort can be avoided in RCD+ using the simple observation that the write buffers need not be flushed—blocks that are logically written but still in the output queue when the distribution finishes, are not flushed to disk but kept in the queues; see also the last sentence in Theorem 5.1. When we read a block in the subsequent recursive sorting phases, we therefore have to check whether this block is still in the output queue and should be taken from there.

Recursive sorting of the classes proceeds depth first, from left to right. As soon as a class fits into internal memory, it is loaded and sorted internally, then it is output using any kind of striping. No randomization is needed for the final output because there is only a single data stream. It suffices to keep $D$ output buffer blocks for the final output. Since the output is generated in sorted order, these output buffers need not be flushed when we are finished with a class which would lead to load imbalance for writing and partially filled blocks. Rather these buffer blocks are kept until they are filled by the sorting of subsequent classes. This way the output is produced in a perfectly striped fashion without partially filled blocks.

**8. A tight lower bound for external sorting.** Our main result for parallel disk sorting is that we close the gap between the upper and lower bounds up to lower-order terms. However, the lower bound from [1] leaves open the constant factors. In particular, it is not clear there what happens if the number of output steps and input steps differ. Therefore we now strengthen the lower bound to obtain the right constant factor.

THEOREM 8.1. *Assuming that $M/B$ is an increasing function, the number of I/Os required to sort or permute $n$ items, up to lower-order terms, is at least*

$$\frac{2N}{D}\frac{\log(N/B)}{B\log(M/B)+2\log N} \sim \begin{cases} \dfrac{2N}{DB}\dfrac{\log(N/B)}{\log(M/B)} & \textit{if } B\log\dfrac{M}{B}=\omega(\log N), \\ \dfrac{N}{D} & \textit{if } B\log\dfrac{M}{B}=o(\log N). \end{cases}$$

The second case in the theorem is the pathological case in which the block size $B$ and internal memory size $M$ are so small that the optimal way to permute the items is to move them one at a time in the naive manner, not making use of blocking.

The rest of this section is devoted to a proof of Theorem 8.1.

For the lower bound calculation, we can assume without loss of generality that there is only one disk, namely, $D = 1$. The I/O lower bound for general $D$ follows by dividing the lower bound for one disk by a factor of $D$.

DEFINITION 8.2. *We call an input operation* simple *if each item that is transferred from the disk gets removed from the disk and deposited into an empty location in internal memory; similarly, an output is* simple *if the transferred items are removed from internal memory and deposited into empty locations on disk.*

LEMMA 8.3 (Aggarwal and Vitter [1]). *For each computation that implements a permutation of the $N$ items, there is a corresponding computation strategy involving only simple I/Os such that the total number of I/Os is no greater.*

*Proof.* It is easy to construct the simple computation strategy by working backwards. We cancel the transfer of an item if its transfer is not needed for the final result. The resulting I/O strategy is simple. ☐

For the lower bound, we use the approach of Aggarwal and Vitter [1] and bound the maximum number of permutations that can be produced by at most $t$ I/Os. If we take the value of $t$ for which the bound first reaches $N!$, we get a lower bound on the worst-case number of I/Os. We can get a lower bound on the average case in a similar way.

DEFINITION 8.4. *We say that a permutation $p_1, p_2, \ldots, p_N$ of the $N$ items can be* produced after $t_I$ input operations and $t_O$ output operations *if there is some intermixed sequence of $t_I$ input operations and $t_O$ output operations so that the items end up in the permuted order $p_1, p_2, \ldots, p_N$ in extended memory. (By extended memory we mean the memory locations of internal memory followed by the memory locations on disk in sequential order.) The items do not have to be in contiguous positions in internal memory or on disk; there can be arbitrarily many empty locations between adjacent items.*

As mentioned above, we assume that I/Os are simple. Each I/O transfers exactly $B$ items, although some of the items may be *nil*. In addition, the I/Os obey block boundaries, in that all the non-*nil* items in a given I/O come from or go to the same block on disk.

Initially, the number of producible permutations is 1. Let us consider the effect of an output. There can be at most $N/B + o - 1$ nonempty blocks before the $o$th output operation, and thus the items in the $o$th output can go into one of $N/B + o$ places relative to the other blocks. Hence, the $o$th output boosts the number of producible permutations by a factor of at most $N/B + o$, which can be bounded trivially by

$$N(1 + \log N). \tag{8.1}$$

For the case of an input operation, we first consider a read I/O from a specific block on disk. If the $b$ items involved in the read I/O were together in internal memory at some previous time (e.g., if the block was created by an earlier output operation), then the items could have been arranged in an arbitrary order by the algorithm while in internal memory. Thus, the $b!$ possible ordering of the $b$ inputted items relative to themselves could already have been produced before the input operation. This implies in a subtle way that rearranging the newly inputted items among the other $M - b$ items in internal memory can boost the number of producible permutations by a factor of at most $\binom{M}{b}$, which is the number of ways to intersperse $b$ indistinguishable items within a group of size $M$.

The above analysis applies to input from a specific block. If the input was preceded by a total of $o$ output operations, there are at most $N/B + o \leq N(1 + \log N)$ blocks to choose from for the I/O, so the number of producible permutations is boosted further by at most $N(1 + \log N)$. Therefore, assuming that at some point the $b$ inputted items were previously together in internal memory, an input operation can boost the number of producible permutations by at most

$$N(1 + \log N)\binom{M}{b}. \tag{8.2}$$

Now let us consider an input operation in which some of the inputted items were not together previously in internal memory (e.g., the first time a block is read). By rearranging the relative order of the items in internal memory, we can increase the number of producible permutations. Given that there are $N/B$ full blocks initially, we get the maximum increase when the $N/B$ blocks are read in full, which boosts the number of producible permutations by a factor of

$$(B!)^{N/B}. \tag{8.3}$$

Let $I$ be the total number of input operations. In the $i$th input operation, let $b_i$ be the number of items brought into internal memory. By the simplicity property, some of the items in the block being accessed may not be brought into internal memory, but rather may be left on disk. In this case, $b_i$ counts only the number of items that are removed from disk and left in internal memory. In particular, we have $0 \leq b_i \leq B$.

By the simplicity property, we need to make room in internal memory for the new items arriving, and in the end all items are stored back on disk. Therefore we get the following lower bound on the number $O$ of output operations:

$$O \geq \frac{1}{B}\left(\sum_{1 \leq i \leq I} b_i\right). \tag{8.4}$$

Combining (8.1), (8.2), and (8.3), we find that

$$\big(N(1+\log N)\big)^{I+O} \prod_{1\le i\le I} \binom{M}{b_i} \ge \frac{N!}{(B!)^{N/B}}, \tag{8.5}$$

where $O$ satisfies (8.4).

Let $\bar{B}$ be the average number of items read during the $I$ input operations. By a convexity argument, the left-hand side of (8.5) is maximized when each $b_i$ has the same value, namely, $\bar{B}$. From (8.5) and (8.4), we get

$$\big(N(1+\log N)\big)^{I+O} \binom{M}{\bar{B}}^{I} \ge \frac{N!}{(B!)^{N/B}}, \tag{8.6}$$

$$\big(N(1+\log N)\big)^{I+O} \binom{M}{\bar{B}}^{(I+O)/(1+\bar{B}/B)} \ge \frac{N!}{(B!)^{N/B}}. \tag{8.7}$$

The left-hand side of (8.7) is maximized when $\bar{B} = B$, so we get

$$\big(N(1+\log N)\big)^{I+O} \binom{M}{B}^{(I+O)/2} \ge \frac{N!}{(B!)^{N/B}}. \tag{8.8}$$

The theorem follows by taking logarithms of both sides of (8.8) and using Stirling's formula and the fact that $M/B$ is an increasing function.

## 9. Conclusions.

In this paper we have exploited a natural duality between prefetching (read problem) and outputting (write problem). We have shown that an optimal schedule for one problem is the reverse of an optimal schedule for the other. We have generalized our approach to the read-many case in which frequently accessed blocks can be cached in memory. We have further reduced the problem of mergesorting and distribution sorting to the read and write problems, and by duality we have given practical yet asymptotically optimal (up to lower-order terms) em algorithms for mergesort and distribution sort. The algorithms are practical [12] and have very low overheads, thus making them desirable in practice.

## Appendix. Summary of notation.

$B$: Block size.
$b_i$: The $i$th block in a sequence of blocks.
$D$: Number of disks. In an acronym it stands for a sorting algorithm based on data Distribution.
$d$: Index of some disk.
disk($b_i$): The disk where block $b_i$ is allocated.
**FR**: Fully random allocation.
iStep($b_i$): The input step when block $b_i$ is fetched.
i$\mathcal{B}$acklog($b_i$): $|\{j > i : \text{iStep}(b_j) \le \text{iStep}(b_i)\}|$.
$L$: Number of blocks in the access sequence $\Sigma$.
$M$: Size of the fast internal memory. In an acronym it stands for a sorting algorithm based on Merging.
$m$: Number of buffer blocks in the buffer pool. Note that $m \le M/B$. In the sorting algorithms $m = \Theta(M/B)$.
$N$: The number of elements to be sorted.
oStep($b_i$): The output step when block $b_i$ is fetched.
o$\mathcal{B}$acklog($b_i$): $|\{j < i : \text{oStep}(b_j) \ge \text{oStep}(b_i)\}|$.

**RC**: Randomized Cycling allocation.
$\pi_i$: In RC allocation the random permutation used to allocate sequence $i$.
**S**: Stands for Striping in an allocation strategy or sorting algorithm.
**SR**: Simple randomized Striping using a random rotation.
Sort($N$): $\frac{N}{DB}(1 + \log_{M/B} \frac{N}{M})$ the I/O complexity of sorting $N$ elements "without the constant factor."
$\Sigma$: The sequence of blocks to be read or written.
$\Sigma^R$: The reverse of sequence $\Sigma$.

**Acknowledgments.** We would like to thank Jeffrey Chase, Andreas Crauser, S. Mitra, Nitin Rajput, Erhard Rahm, and Berthold Vöcking for valuable discussions. Anonymous referees have given detailed feedback that helped to improve the readability of the paper.

## REFERENCES

[1] A. Aggarwal and J. S. Vitter, *The input/output complexity of sorting and related problems*, Comm. ACM, 31 (1988), pp. 1116–1127.

[2] S. Albers, *On the influence of lookahead in competitive paging algorithms*, Algorithmica, 18 (1997), pp. 283–305.

[3] S. Albers and M. Büttner, *Integrated prefetching and caching in single and parallel disk systems*, in Proceedings of the 15th Annual ACM Symposium on Parallel Algorithms and Architectures, San Diego, 2003, pp. 109–117.

[4] S. Albers, N. Garg, and S. Leonardi, *Minimizing stall time in single and parallel disk systems*, in Proceedings of the 30th Annual ACM Symposium on Theory of Computing (STOC-98), New York, 1998, ACM Press, pp. 454–462.

[5] R. D. Barve, E. F. Grove, and J. S. Vitter, *Simple randomized mergesort on parallel disks*, Parallel Comput., 23 (1997), pp. 601–631.

[6] R. D. Barve and J. S. Vitter, *A simple and efficient parallel disk mergesort*, in Proceedings of the 11th Annual ACM Symposium on Parallel Algorithms and Architectures, St. Malo, France, 1999, pp. 232–241.

[7] A. L. Belady, *A study of replacement algorithms for virtual storage computers*, IBM Systems J., 5 (1966), pp. 78–101.

[8] A. Borodin and R. El-Yaniv, *Online Computation and Competitive Analysis*, Cambridge University Press, Cambridge, UK, 1998.

[9] P. Cao, E. W. Felten, A. R. Karlin, and K. Li, *Implementation and performance of integrated application-controlled file caching, prefetching, and disk scheduling*, ACM Trans. Comput. Systems, 14 (1996), pp. 311–343.

[10] F. Dehne, W. Dittrich, and D. Hutchinson, *Efficient external memory algorithms by simulating coarse-grained parallel algorithms*, in Proceedings of the 9th Annual ACM Symposium on Parallel Algorithms and Architectures, Newport, RI, 1997, pp. 106–115.

[11] F. Dehne, W. Dittrich, D. Hutchinson, and A. Maheshwari, *Reducing I/O complexity by simulating coarse grained parallel algorithms*, in proceedings of the 13th Annual International Parallel Processing Symposium, IEEE, San Juan, Puerto Rico, 1999, pp. 14–20.

[12] R. Dementiev and P. Sanders, *Asynchronous parallel disk sorting*, in Proceedings of the 15th Annual ACM Symposium on Parallelism in Algorithms and Architectures, San Diego, 2003, pp. 138–148.

[13] D. A. Hutchinson, P. Sanders, and J. S. Vitter, *Duality between prefetching and queued writing with parallel disks*, in Proceedings of the 9th Annual European Symposium on Algorithms (ESA), Lecture Notes in Comput. Sci. 2161, Springer, Berlin, 2001, pp. 62–73.

[14] D. A. Hutchinson, P. Sanders, and J. S. Vitter, *The power of duality for prefetching and sorting with parallel disks*, in Proceedings of the 12th Annual ACM Symposium on Parallel Algorithms and Architectures (SPAA Revue), Crete Island, Greece, 2001, pp. 334–335.

[15] M. Kallahalla and P. J. Varman, *Optimal read-once parallel disk scheduling*, in Proceedings of the 6th Annual Workshop on Input/Output in Parallel and Distributed Systems, Atlanta, GA, 1999, pp. 68–77.

[16] M. Kallahalla and P. J. Varman, *Optimal prefetching and caching for parallel I/O systems*, in Proceedings of the 13th Annual Symposium on Parallel Algorithms and Architectures, Crete Island, Greece, 2001, pp. 219–228.

[17] M. Kallahalla and P. J. Varman, *PC-OPT: Optimal offline prefetching and caching for parallel I/O systems*, IEEE Trans. Comput., 51 (2002), pp. 1333–1344.

[18] T. Kimbrel and A. R. Karlin, *Near-optimal parallel prefetching and caching*, SIAM J. Comput., 29 (2000), pp. 1051–1082.

[19] D. E. Knuth, *The Art of Computer Programming—Sorting and Searching*, Vol. 3, 2nd ed., Addison Wesley, Reading, MA, 1998.

[20] M. H. Nodine and J. S. Vitter, *Deterministic distribution sort in shared and distributed memory multiprocessors*, in Proceedings of the 5th Annual ACM Symposium on Parallel Algorithms and Architectures, Velen, Germany, 1993, pp. 120–129.

[21] M. H. Nodine and J. S. Vitter, *Greed sort: Optimal deterministic sorting on parallel disks*, J. ACM, 42 (1995), pp. 919–933.

[22] P. Sanders, S. Egner, and J. Korst, *Fast concurrent access to parallel disks*, in Proceedings of the 11th Annual ACM-SIAM Symposium on Discrete Algorithms, San Francisco, 2000, pp. 849–858.

[23] J. S. Vitter, *External memory algorithms and data structures: Dealing with MASSIVE data*, ACM Computing Surveys, 33 (2001), pp. 209–271.

[24] J. S. Vitter and D. A. Hutchinson, *Distribution sort with randomized cycling*, in Proceedings of the 12th Annual ACM-SIAM Symposium on Discrete Algorithms, Washington, D.C., 2001, ACM, New York, SIAM, Philadelphia, pp. 77–86.

[25] J. S. Vitter and E. A. M. Shriver, *Algorithms for parallel memory* I. *Two-level memories*, Algorithmica, 12 (1994), pp. 110–147.

SIAM J. COMPUT.
Vol. 34, No. 6, pp. 1464–1473

# DECIDABLE AND UNDECIDABLE PROBLEMS ABOUT QUANTUM AUTOMATA[*]

VINCENT D. BLONDEL[†], EMMANUEL JEANDEL[‡], PASCAL KOIRAN[‡], AND NATACHA PORTIER[‡]

**Abstract.** We study the following decision problem: is the language recognized by a quantum finite automaton empty or nonempty? We prove that this problem is decidable or undecidable depending on whether recognition is defined by strict or nonstrict thresholds. This result is in contrast with the corresponding situation for probabilistic finite automata, for which it is known that strict and nonstrict thresholds both lead to undecidable problems.

**Key words.** quantum automata, probabilistic automata, undecidable problems, algebraic groups

**AMS subject classifications.** 81P68, 68Q45

**DOI.** 10.1137/S0097539703425861

## 1. Introduction.

In this paper, we provide decidability and undecidability proofs for two problems associated with quantum finite automata. Quantum finite automata (QFA) were introduced by Moore and Crutchfield [MC00]; they are to quantum computers what finite automata are to Turing machines. Quantum automata are also analogous to the probabilistic finite automata introduced in the 1960s by Rabin that accept words with a certain probability (see [Rab63], [Rab67]; see also [Paz71] for a book-length treatment). A quantum automaton $A$ assigns real values $\mathrm{Val}_A(w)$ to input words $w$ (see below for a precise description of how these values are computed). $\mathrm{Val}_A(w)$ can be interpreted as the probability that on any given run of $A$ on the input word $w$, $w$ is accepted by $A$. Nonisolated cut-point recognition will be considered in this article: we do not ask for a gap between the set of $\mathrm{Val}_A(w)$ for accepted words $w$ and the set of $\mathrm{Val}_A(w)$ for rejected words $w$. Associated to a real threshold $\lambda$, the languages recognized by the automaton $A$ with nonstrict and strict threshold $\lambda$ are

$$L_{\geq} = \{w : \mathrm{Val}_A(w) \geq \lambda\} \quad \text{and} \quad L_{>} = \{w : \mathrm{Val}_A(w) > \lambda\}.$$

Many properties of these languages are known in the case of probabilistic and quantum automata. For instance, it is known that the class of languages recognized by quantum automata is strictly contained in the class of languages recognized by probabilistic finite automata [BP02]. For probabilistic automata it is also known that the problem of determining if $L_{\geq}$ is empty and the problem of determining if $L_{>}$ is empty are undecidable (see [Paz71, Thm. 6.17, p. 90]). This is true even for automata of fixed dimensions [BC03]. Decidability problems on QFA were first studied in the paper by Amano and Iwama [AI99]: is the language recognized by a 1.5-way quantum automaton empty? The undecidability of this problem was proven, even in the case of isolated cut-point.

[*]Received by the editors April 9, 2003; accepted for publication (in revised form) December 11, 2004; published electronically August 17, 2005.
http://www.siam.org/journals/sicomp/34-6/42586.html

[†]Department of Mathematical Engineering, Université Catholique de Louvain (blondel@inma.ucl.ac.be).
[‡]Laboratoire de l'Informatique du Parallélisme, Ecole Normale Supérieure de Lyon (Emmanuel.Jeandel@ens-lyon.fr, Pascal.Koiran@ens-lyon.fr, Natacha.Portier@ens-lyon.fr).

Table 1
*Decidable and undecidable problems for probabilistic and quantum automata.*

| | $L_{\geq} = \emptyset$ | $L_{>} = \emptyset$ | $L_{\leq} = \emptyset$ | $L_{<} = \emptyset$ |
|---|---|---|---|---|
| PFA | undecidable | undecidable | undecidable | undecidable |
| QFA | undecidable | decidable | undecidable | decidable |

In this contribution, we consider the problem of determining for a quantum automaton $A$ and threshold $\lambda$ if there exists a word $w$ for which $\mathrm{Val}_A(w) \geq \lambda$ and if there exists a word $w$ for which $\mathrm{Val}_A(w) > \lambda$. We prove in Theorem 2.1 and Corollary 2.2 that the first problem is undecidable, and in Theorem 3.1 that the second problem is decidable. For quantum automata it thus makes a difference to consider strict or nonstrict thresholds. This result is in contrast with probabilistic automata, for which both problems are undecidable.

Similarly to the languages $L_{\geq}$ and $L_{>}$, one can define the languages $L_{\leq}$ and $L_{<}$ and ask whether or not they are empty (of course, emptiness of $L_{\leq}$ is equivalent to $L_{>}$ being equal to $\Sigma^*$). These two problems are known [Paz71] to be undecidable for probabilistic automata. For quantum automata our decidability results do again differ depending on whether we consider strict or nonstrict inequalities. Our results are summarized in Table 1.

Before we proceed with the proofs, we first define what we mean by a QFA. A number of different quantum automata models have been proposed in the literature and not all models are computationally equivalent. For the "measure-many" model of quantum automata introduced by Kondacs and Watrous [KW97] the four problems of Table 1 are proven undecidable in [Jea02]. The model we consider here is the so-called measure once quantum finite automaton introduced by Moore and Crutchfield [MC00]. These automata operate as follows. Let $\Sigma$ be a finite set of input letters and let $\Sigma^*$ denote the set of finite input words (including the empty word); typical elements of $\Sigma^*$ will be denoted $w = w_1 \cdots w_{|w|}$, where $w_i \in \Sigma$ and $|w|$ denotes the length of $w$. The QFA $A$ is given by a finite set of $n$ states, $n \times n$ unitary transition matrices $X_\alpha$ (one for each symbol $\alpha$ in $\Sigma$), a (row) vector of unit norm $s$ (the initial configuration), and an $n \times n$ orthogonal projection matrix $P$. Given a word $w \in \Sigma^*$, the value of $w$, denoted $\mathrm{Val}_A(w)$, is defined by

$$\mathrm{Val}_A(w) = \|sX_wP\|^2.$$

In this expression, $\|\cdot\|$ is the euclidean vector norm, and we use the notation $X_w$ for the product $X_{w_1} \cdots X_{w_{|w|}}$. For a vector $v$, the value $\|vP\|^2$ is the probability for the quantum state $v$ to be observed in acceptance space. The value $\mathrm{Val}_A(w)$ can thus be interpreted as the probability of observing the quantum state in acceptance space after having applied the operator sequence $X_{w_1}$ to $X_{w_{|w|}}$ to the initial quantum state $s$.

The rest of the paper is organized as follows. In section 2, we reduce Post's correspondence problem to the problem of determining if a quantum automata has a word of value larger than or equal to a given threshold. Post's correspondence problem is undecidable, and this therefore proves our first result. Our reduction uses an encoding of words in three-dimensional space. In section 3, we prove decidability of the same problem for strict inequality. For the proof we use the fact that any compact matrix group is algebraic, and the group we consider can be given an effective description.

**Complex versus real entries.** Throughout the paper we will assume that the initial state, the unitary matrices $X_\alpha$, and the projection matrix $P$ have real rather than complex entries (i.e., these matrices are actually orthogonal). This is not a significant restriction since any quantum automaton $A$ (with possibly complex entries) can be simulated by another quantum automaton $A'$ with real entries by doubling the number of states. More precisely, let $Q$ be the set of states of $A$. We replace each element $q_j$ of $Q$ by two states $q_j^1$ and $q_j^2$. Let $\phi : \mathbb{C}^n \to \mathbb{R}^{2n}$ be the $\mathbb{R}$-linear map which sends a configuration $x = \sum_{j=1}^n (\alpha_j + i\beta_j) q_j$ to $\phi(x) = \sum_{j=1}^n \alpha_j q_j^1 + \sum_{j=1}^n \beta_j q_j^2$. We replace the initial configuration $s$ by $s' = \phi(s)$. Let $X$ be one of the matrices of $A$. The rows and columns of $A$ are indexed by elements of $Q$. Let $x_{jk} + iy_{jk}$ be the entry at row $q_j$ and column $q_k$. Recall that a complex number $x + iy$ can be identified to the 2 × 2 matrix

$$\begin{pmatrix} x & -y \\ y & x \end{pmatrix} .$$

It is therefore natural to replace this entry by the 2 × 2 matrix

$$\begin{pmatrix} x_{jk} & -y_{jk} \\ y_{jk} & x_{jk} \end{pmatrix} .$$

The two rows and two columns of this matrix are indexed, respectively, by $q_j^1, q_j^2, q_k^1$, and $q_k^2$. By abuse of notation we also denote by $\phi$ the map which sends $X$ to $X'$. It is easy but instructive to check that for any $v \in \mathbb{C}^n$ and for any $n \times n$ complex matrices $A$ and $B$ the following relations hold: $\phi(Av) = \phi(A)\phi(v)$, $\phi(AB) = \phi(A)\phi(B)$, $\phi(A^*) = \phi(A)^T$, and $v^*v = \phi(v)^T\phi(v)$. Now recall that unitary matrices, orthogonal matrices, complex matrices of orthogonal projection, and real matrices of orthogonal projection are, respectively, characterized by the following relations: $AA^* = I$, $AA^T = I$, $A = A^* = A^2$, and $A = A^T = A^2$. It follows that $\phi$ sends unitary matrices to orthogonal matrices, and complex matrices of orthogonal projection to real matrices of orthogonal projection. The quantum automaton $A'$ defined by the orthogonal matrices $X'_a = \phi(X_a)$, the projection matrix $P' = \phi(P)$, and the initial configuration $s'$ satisfies $\phi(sX_wP) = s'X'_wP'$ for any word $w$. Hence $\mathrm{Val}_A(w) = \mathrm{Val}_{A'}(w)$ for any word $w$.

## 2. Undecidability for nonstrict inequality.

We prove in this section that the problem of determining if a quantum automata has a word of value larger than or equal to some threshold is undecidable. The proof is by reduction from Post's correspondence problem (PCP), a well-known undecidable problem. An instance of PCP is given by a finite alphabet $\Sigma$ and $k$ pairs of words $(u_i, v_i) \in \Sigma^* \times \Sigma^*$ for $i = 1, \dots, k$. A solution to the correspondence is any nonempty word $w = w_1 \cdots w_n$ over the alphabet $\{1, \dots, k\}$ such that $u_w = v_w$, where $u_w = u_{w_1} \dots u_{w_n}$. This correspondence problem is known to be undecidable: there is no algorithm that decides if a given instance has a solution [Pos46]. It is easy to see that the problem remains undecidable when the alphabet $\Sigma$ contains only two letters. The problem is also known to be undecidable for $k = 7$ pairs [MS05] but is decidable for $k = 2$ pairs; the decidability of the cases $2 < k < 7$ is not yet known. We are now ready to state our first result.

THEOREM 2.1. *There is no algorithm that decides for a given automaton $A$ if there exists a nonempty word $w$ for which $\mathrm{Val}_A(w) \le 0$, or if there exists one for which $\mathrm{Val}_A(w) \ge 1$. These two problems remain undecidable even if the automaton is given by 7 orthogonal matrices in dimension 6.*

*Proof.* We proceed by reduction from PCP. For our reduction we need to encode words by orthogonal matrices. We will take matrices that represent rotations of angle arccos(3/5) on, respectively, the first and third axes:

$$X_a = \frac{1}{5}\begin{pmatrix}3 & -4 & 0\\ 4 & 3 & 0\\ 0 & 0 & 5\end{pmatrix}, \quad X_b = \frac{1}{5}\begin{pmatrix}5 & 0 & 0\\ 0 & 3 & -4\\ 0 & 4 & 3\end{pmatrix}.$$

These matrices are orthogonal, $X_a X_a^T = I = X_b X_b^T$, and they generate a free group since a result from Swierczkowski [Sw58, Sw94] ensures that if $\cos\phi \in \mathbb{Q}$, two rotations of angle $\phi$ on orthogonal axes in $\mathbb{R}^3$ generate a free group if and only if $\cos\phi \notin \{0, \pm\frac{1}{2}, \pm 1\}$.

In addition to that, we now prove that there exists a vector $t$ such that $tX_u = tX_w$ implies $u = w$.

We will use here a method from [Su90]. One can show by induction that for any reduced matrix product $M$ of $k$ matrices[1] taken from the set $\{X_a, X_b, X_a^{-1}, X_b^{-1}\}$, we have

$$(3\ 0\ 4)M = (x_1\ x_2\ x_3)/5^k$$

with $x_1, x_2, x_3 \in \mathbb{Z}$, and 5 divides $x_2$ if and only if $k = 0$ (and then $M = I$).

The result is obviously true for $k = 0, 1$. Now, if $M = M' X_a X_b$, then $(3\ 0\ 4)M = (x_1\ x_2\ x_3)/5^k X_a X_b = (x_4\ x_5\ 5x_3)/5^{k+1} X_b$ for some $x_4, x_5$, and by induction hypothesis, 5 does not divide $x_5$. Now $(3\ 0\ 4)M = (x_6\ 3x_5 + 20x_3\ x_7)/5^{k+2}$ so that 5 does not divide the second term. The proofs for all the other cases are similar.

We will now call $t$ the row vector $(3\ 0\ 4)$. If $tX_u = tX_v$, then $tX_u X_v^{-1} = t$. As the second component of $t$ is equal to 0, the product must be trivial, and so $u = v$.

Given an instance $(u_i, v_i)_{1 \le i \le k}$ of PCP over the alphabet $\{a, b\}$ and a word $w \in \{1, \ldots, k\}^*$, we construct the matrix

$$Y_w = \frac{1}{2}\begin{pmatrix}X_{u_w} + X_{v_w} & X_{u_w} - X_{v_w}\\ X_{u_w} - X_{v_w} & X_{v_w} + X_{u_w}\end{pmatrix}.$$

These matrices are orthogonal and verify $Y_{w\nu} = Y_w Y_\nu$.

A solution of the original PCP problem is a nonempty word $w \in \{1, \ldots, k\}^*$ such that the upper-right block of the matrix $Y_w$ is equal to zero. We may use the previously introduced vector $t = (3\ 0\ 4)$ to test this condition. We have

$$\begin{pmatrix}t & 0\end{pmatrix} Y_w = \frac{1}{2}\begin{pmatrix}tX_{u_w} + tX_{v_w} & tX_{u_w} - tX_{v_w}\end{pmatrix},$$

and thus a solution of the PCP problem is a word $w$ such that the last three coordinates of $yY_w$ are equal to zero, where $y = \begin{pmatrix}t & 0\end{pmatrix}$. This condition can be tested with a projection matrix. Defining

$$P = \begin{pmatrix}0_3 & 0\\ 0 & I_3\end{pmatrix}$$

[1]A product is said to be *reduced* if no two consecutive matrices in the product are inverse from each other.

we have that the solutions of the original PCP problem are the words $w$ for which $y\ Y_w\ P = 0$, which is equivalent to

$$\mathrm{Val}_A(w) = \|yY_wP\|^2 = 0.$$

The values taken by $\mathrm{Val}_A(w)$ are nonnegative and so the problem of determining if there exists a nonempty word $w$ such that $\mathrm{Val}_A(w) \leq 0$ is undecidable. Notice also that $\|yY_wI\|^2 = 1$ and so

$$\|yY_w(I-P)\|^2 \leq 1$$

with equality only for $yY_wP = 0$. Thus, the problem of determining if there exists a nonempty word $w$ such that $\mathrm{Val}_A(w) \geq 1$ is undecidable too. ∎

Theorem 2.1 deals only with nonempty words. We remove this restriction in the next result, and we reduce the number of matrices from 7 to 2.

COROLLARY 2.2. *There is no algorithm that decides for a given automaton $A$ if there exists a word $w$ for which $\mathrm{Val}_A(w) \leq 0$, or if there exists one for which $\mathrm{Val}_A(w) \geq 1$. These problems remain undecidable even if the automaton is given by 7 orthogonal matrices in dimension 6, or by 2 orthogonal matrices in dimension 42.*

*Proof.* As in the proof of Theorem 2.1, the undecidability results for the condition $\mathrm{Val}_A(w) \geq 1$ follow from those for the condition $\mathrm{Val}_A(w) \leq 0$. Hence we supply the proofs for the latter condition only. We proceed by reduction from the problem $\exists w\ \mathrm{Val}_A(w) \leq 0$ for 7 matrices in dimension 6, which is undecidable for nonempty words $w$ as shown in Theorem 2.1. Note that the language of the nonempty $w$'s such that $\mathrm{Val}_A(w) \leq 0$ is the union of the seven languages defined by the conditions $\mathrm{Val}_A(iw) \leq 0$ for possibly empty words $w$ and $i \in \{1, \ldots, 7\}$. Hence the emptiness of one of these languages (say, the first one) must be undecidable. Thus, the problem of determining if there exists a word $w$ such that $\mathrm{Val}_A(1w) \leq 0$ is undecidable.[2] For each automaton $A = ((Y_i)_{i\in\{1,\ldots,7\}}, s, P)$ we can now construct the quantum automaton $B = ((Y_i)_{i\in\{1,\ldots,7\}}, y, P)$, where $y = sY_1$. Then $\mathrm{Val}_A(1w) \leq 0$ if and only if $\mathrm{Val}_B(w) \leq 0$.

The following problem is therefore undecidable: given a quantum automaton $A$ defined by 7 orthogonal matrices in dimension 6, is there a (possibly empty) word $w$ such that $\mathrm{Val}_A(w) \leq 0$?

Finally, we show how to reduce the number of matrices to 2. We use a construction from Blondel and Tsitsiklis [BT97] and Blondel and Caterini [BC03]. Given the above matrices $Y_i$ and the projection matrix $P$, we define

$$Z_0 = \begin{pmatrix} Y_1 & 0 & \ldots & 0 \\ 0 & Y_2 & \ddots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \ldots & Y_7 \end{pmatrix} \quad \text{and} \quad Z_1 = \begin{pmatrix} 0 & I & 0 & 0 \\ 0 & \ddots & \ddots & 0 \\ \vdots & \vdots & \ddots & I \\ I & 0 & \ldots & 0 \end{pmatrix}.$$

When taking products of these two matrices the matrix $Z_1$ acts as a "selecting matrix" on the blocks of $Z_0$. Let us define $x = \begin{pmatrix} y & 0 \end{pmatrix}$ and

$$Q = \begin{pmatrix} P & 0 & \ldots & 0 \\ 0 & P & \ddots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \ldots & P \end{pmatrix}.$$

[2]It is not difficult to show that the 6 other problems must be undecidable as well.

We claim that there exists a word $w$ over the alphabet $\{1,\dots,7\}$ such that $\|yY_wP\| = 0$ if and only if there exists a word $\nu$ over $\{0,1\}$ such that $\|xZ_\nu Q\| = 0$. Indeed, for any word $\nu$ over $\{0,1\}$, $xZ_\nu$ is a row vector of block form $(0\cdots 0\ yY_w\ 0\cdots 0)$ for some word $w$ over $\{1,\dots,7\}$ (the length of $w$ is equal to the number of 0's in $\nu$). Therefore $\|xZ_\nu Q\| = \|yY_wP\|$. Conversely, for any word $w$ over $\{1,\dots,7\}$ there exists a word $\nu$ over $\{0,1\}$ such that $xZ_\nu$ is a row vector of block form $(yY_w\ 0\cdots 0)$, and we therefore have again the equality $\|xZ_\nu Q\| = \|yY_wP\|$. To obtain $Z_\nu$ from $Y_w$, one can for instance replace as in [BT97] each matrix $Y_i$ in the product $Y_w$ by $Z_1^{-(8-i)}Z_0Z_1^{(8-i)} = Z_1^{i-1}Z_0Z_1^{8-i}$. ∎

Theorem 2.1 and its corollary deal only with 0/1 thresholds. We prove below that, whichever threshold $0 < \lambda \leq 1$ is used, the problem of determining if there exists a word for which $\mathrm{Val}_A(w) \geq \lambda$ is undecidable. This result follows as a corollary to the following lemma.

LEMMA 2.3. *Associated to every QFA $A$ and threshold $0 < \lambda \leq 1$ we can effectively construct a QFA $B$ such that the language recognized with threshold $\lambda$ by $B$ is the language recognized with threshold 1 by $A$. Moreover, if $\lambda \in \mathbb{Q}$ and $A$ has only rational entries, then $B$ can be chosen with rational entries.*

*Proof.* The idea is to construct $B$ by adding a state to $A$. Let $A$ be given by the orthogonal matrices $X_i^A$, the projection matrix $P^A$, and the initial vector $s^A$. Let

$$X_i^B = \begin{pmatrix} X_i^A & 0 \\ 0 & 1 \end{pmatrix},$$

and define $s^B = \begin{pmatrix} \sqrt{\lambda}\, s^A & \sqrt{1-\lambda} \end{pmatrix}$. If we choose

$$P^B = \begin{pmatrix} P^A & 0 \\ 0 & 0 \end{pmatrix},$$

we immediately have $\mathrm{Val}_B(w) = \lambda\ \mathrm{Val}_A(w)$ and the first part of lemma is proven. The entries $\sqrt{\lambda}$ and $\sqrt{1-\lambda}$ in general do not need to be rational. It remains to show how the parameters of $B$ can be chosen rational when those of $A$ are. We therefore use Lagrange's theorem to write $\lambda$ and $1-\lambda$ as the sum of the squares of four rational numbers, say $\lambda = a_1^2 + a_2^2 + a_3^2 + a_4^2$ and $1 - \lambda = b_1^2 + b_2^2 + b_3^2 + b_4^2$.

Now, if we define

$$s^B = \begin{pmatrix} a_1 s^A & a_2 \cdots a_4 & b_1 \cdots b_4 \end{pmatrix}\ X_i^B = \begin{pmatrix} X_i^A & 0 \\ 0 & I_7 \end{pmatrix}\ P^B = \begin{pmatrix} P^A & 0 & 0 \\ 0 & I_3 & 0 \\ 0 & 0 & 0_4 \end{pmatrix},$$

we immediately have $\mathrm{Val}_B(w) = a_1^2\mathrm{Val}_A(w) + a_2^2 + a_3^2 + a_4^2$, $\|s^B\|^2 = 1$ and the lemma is proven. ∎

Combining Lemma 2.3 with Corollary 2.2, we immediately obtain the following.

COROLLARY 2.4. *For any rational $\lambda$, $0 < \lambda \leq 1$, there is no algorithm that decides if a given quantum automata has a word $w$ for which* $\mathrm{Val}(w) \geq \lambda$.

## 3. Decidability for strict inequality.

We now prove that the problem of determining if a quantum automata has a word of value *strictly* larger than some threshold is decidable. This result points to a difference between quantum and probabilistic automata since for probabilistic automata this problem is known to be undecidable.

Once an automaton is given, one can of course always enumerate all possible words $w$ and halt as soon as one is found for which $\mathrm{Val}_A(w) > \lambda$, and so the problem is clearly semidecidable. In order to show that it is decidable, it remains to exhibit a procedure that halts when $\mathrm{Val}_A(w) \leq \lambda$ for all $w$.

Let a quantum automata $A$ be given by a finite set of $n \times n$ orthogonal transition matrices $X_i$, an initial configuration $s$ of unit norm, and a projection matrix $P$. The value of the word $w$ is given by $\mathrm{Val}_A(w) = \|sX_wP\|^2$. Let $\mathcal{X}$ be the semigroup generated by the matrices $X_i$, $\mathcal{X} = \{X_w : w \in \Sigma^*\}$, and let $f : \mathbb{R}^{n\times n} \mapsto \mathbb{R}$ be the function defined by $f(X) = \|sXP\|^2$. We have that

$$\mathrm{Val}_A(w) = f(X_w),$$

and the problem is now that of determining if $f(X) \leq \lambda$ for all $X \in \mathcal{X}$. The function $f$ is a (continuous) polynomial map and so this condition is equivalent to $f(X) \leq \lambda$ for all $X \in \overline{\mathcal{X}}$, where $\overline{\mathcal{X}}$ is the closure of $\mathcal{X}$ in $\mathbb{R}^{n\times n}$. The set $\overline{\mathcal{X}}$ has the interesting property that it is algebraic (see below for a proof), and so there exist polynomial mappings $f_1, \dots, f_p : \mathbb{R}^{n\times n} \mapsto \mathbb{R}$, such that $\overline{\mathcal{X}}$ is exactly the set of common zeros of $f_1, \dots, f_p$. If the polynomials $f_1, \dots, f_p$ are known, the problem of determining whether $f(X) \leq \lambda$ for all $X \in \overline{\mathcal{X}}$ can be written as a quantifier elimination problem

$$\forall X \big[ (f_1(X) = 0 \wedge \cdots \wedge f_p(X) = 0) \implies f(X) \leq \lambda \big]. \tag{3.1}$$

This is a first-order formula over the reals and can be decided effectively by Tarski–Seidenberg elimination methods (see [Ren92a, Ren92b, Ren92c, BPR96] for a survey of known algorithms). If we knew how to effectively compute the polynomials $f_1, \dots, f_p$ from the matrices $X_i$, a decision algorithm would therefore follow immediately. In the following we solve a simpler problem: we effectively compute a sequence of polynomials whose zeros describe the same set $\overline{\mathcal{X}}$ after finitely many terms (but we may never know how many). It turns out that this is sufficient for our purposes. We will use some basic algebraic geometry. In particular, we will use the Noether (or "descending chain") property: in any field, the set of common zeros of a set of $n$-variate polynomials is equal to the set of common zeros of a *finite* subset of these polynomials (see any textbook on algebraic geometry, for instance, [CLO92, Prop. 1, sect. 4.6]).

THEOREM 3.1. *Let $(X_i)_{i\in\Sigma}$ be orthogonal matrices of dimension $n$ and let $\overline{\mathcal{X}}$ be the closure of the semigroup $\{X_w : w \in \Sigma^*\}$. The set $\overline{\mathcal{X}}$ is algebraic, and if the $X_i$ have rational entries, we can effectively compute a sequence of polynomials $f_1, \dots, f_i, \dots$ such that*

1. *if $X \in \overline{\mathcal{X}}$, $f_i(X) = 0$ for all $i$;*
2. *there exists some $k$ such that $\overline{\mathcal{X}} = \{X : f_i(X) = 0, i = 1, \dots, k\}$.*

*Proof.* We first prove that $\overline{\mathcal{X}}$ is algebraic. It is known (see, e.g., [OV90]) that every compact group of real matrices is algebraic. In fact, the proof of algebraicity in [OV90] reveals that any compact group $G$ of real matrices of size $n$ is the zero set of

$$\mathbb{R}[X]^G = \{f \in \mathbb{R}[X] : f(I) = 0 \text{ and } f(gX) = f(X)\, \forall\, g \text{ in } G\};$$

i.e., $G$ is the zero set of the polynomials in $n \times n$ variables which vanish at the identity and are invariant under the action of $G$. We will use this property later in the proof.

To show that $\overline{\mathcal{X}}$ is algebraic, it suffices to show that $\overline{\mathcal{X}}$ is compact and is a group. The set $\overline{\mathcal{X}}$ is obviously compact (bounded and closed in a normed vector space of finite dimension). Let us show that it is a group. It is in fact known that every compact subsemigroup of a topological group is a subgroup. Here is a self-contained proof in our setting: For every matrix $X$, the sequence $X^k$ admits a subsequence that is a

Cauchy sequence, by compactness. Hence for every $\epsilon$ there exists $k > 0$ and $l > k+1$ such that $\|X^k - X^l\| \leq \epsilon$, that is, $\|X^{-1} - X^{l-k-1}\| \leq \epsilon$ (recall that $\|AB\| = \|B\|$ if $A$ is orthogonal and if $\|.\|$ is the operator norm associated to the euclidean norm). Hence, $X^{-1}$ is in the set and the first part of the theorem is proven. For notational convenience, we will denote the group $\overline{\mathcal{X}}$ by $G$ in the remainder of the proof.

For the second part of the theorem, we will prove that we can take

$$\{f_i\} = \{f \in \mathbb{Q}[X] : f(I) = 0 \text{ and } f(X_j X) = f(X)\, \forall j \text{ in } \Sigma\}.$$

In other words, this is the set $\mathbb{Q}[X]^G$ of rational polynomials which vanish at the identity and are invariant under the action of each matrix $X_j$. It is clear that this set is recursively enumerable. We claim that $G$ is the zero set of the $f_i$'s. By Noetherianity the zero set of the $f_i$'s is equal to the zero set of a finite subset of the $f_i$'s, so that the theorem follows immediately from this claim. To prove the claim, we will use the fact that $G$ is the zero set of $\mathbb{R}[X]^G$. Note that

$$\mathbb{R}[X]^G = \{f \in \mathbb{R}[X] : f(I) = 0 \text{ and } f(X_j X) = f(X)\, \forall j \text{ in } \Sigma\}.$$

(A polynomial is invariant under the action of $G$ if and only if it is invariant under the action of all the $X_j$.) This observation implies immediately that each $f_i$ is in $\mathbb{R}[X]^G$, so that the zero set of the $f_i$'s contains the zero set of $\mathbb{R}[X]^G$. The converse inclusion follows from the fact that any element $P$ of $\mathbb{R}[X]^G$ can be written as a linear combination of some $f_i$'s. Indeed, let $d$ be the degree of $P$ and let $E_d$ be the set of real polynomials in $n \times n$ variables of degree at most $d$. The set $V_d = E_d \cap \mathbb{R}[X]^G$ is a linear subspace of $E_d$ defined by a system of linear equations with rational coefficients (those equations are $f(I) = 0$ and $f(X_j X) = f(X)$ for all $j \in \Sigma$). Hence there exists a basis of $V_d$ made up of polynomials with rational coefficients, that is, of elements of $\{f_i\}$. This completes the proof of the claim, and of the theorem. $\square$

We may now apply this result to quantum automata.

THEOREM 3.2. *The two following problems are decidable:*

(i) *Given a quantum automaton $A$ and a threshold $\lambda$, decide whether there exists a word $w$ such that* $\mathrm{Val}_A(w) > \lambda$.

(ii) *Given a quantum automaton $A$ and a threshold $\lambda$, decide whether there exists a word $w$ such that* $\mathrm{Val}_A(w) < \lambda$.

*Proof.* We show only that problem (i) is decidable. The argument for problem (ii) is essentially the same.

As pointed out at the beginning of this section, it suffices to exhibit an algorithm which halts if and only if $\mathrm{Val}_A(w) \leq \lambda$ for every word $w$. Consider the following algorithm:

- enumerate the $f_i$'s;
- for every initial segment $f_1, \ldots, f_p$, decide whether (3.1) holds, and halt if it does.

It follows from property (1) in Theorem 3.1 that $\mathrm{Val}_A(w) \leq \lambda$ for every word $w$ if the algorithm halts. The converse follows from property (2). $\square$

In Theorems 3.1 and 3.2 we have assumed that our orthogonal matrices have rational entries, mostly because the undecidability results of section 2 already hold for rational entries. It is not hard to relax this hypothesis. For instance, it is clear from the proofs that Theorems 3.1 and 3.2 can be generalized to matrices with real algebraic entries. Even more generally, one may allow "arbitrary" real entries by proceeding as follows. Let $K$ be the subfield of $\mathbb{R}$ generated by the entries of our matrices. We may give a transcendence basis $\mathcal{B}$ of $K$ and represent the entries as algebraic numbers over $\mathcal{B}$. This purely algebraic information is sufficient to compute the sequence of polynomials $(f_i)$ in Theorem 3.1. We also need to decide for every initial segment whether (3.1) holds. After quantifier elimination, this amounts to computing the sign of a finite number of polynomial functions of the elements of $\mathcal{B}$. In order to do this we need only assume that we have access to an oracle which for any element $x$ of $\mathcal{B}$ and any $\epsilon > 0$ outputs a rational number $q$ such that $|x - q| < \epsilon$ (such an oracle can be effectively implemented if the entries are computable real numbers). We use the algebraic information to determine whether a polynomial takes the value zero, and if not we use approximations to determine its sign.

In the proof of Theorem 3.2 we have bypassed the problem of explicitly computing a finite set of polynomials defining $\overline{\mathcal{X}}$. It is in fact possible to show that this problem is algorithmically solvable [DJK03]. This implies in particular that the following two problems are decidable:

(i) Decide whether a given threshold is isolated.

(ii) Decide whether a given QFA has an isolated threshold.

A threshold $\lambda$ is said to be isolated if

$$\exists \epsilon > 0 \; \forall w \in \Sigma^* \; |\mathrm{Val}_A(w) - \lambda| > \epsilon.$$

It is known that these two problems are undecidable for probabilistic automata [Ber75, BMT77, BC03].

The algorithm of [DJK03] for computing $\overline{\mathcal{X}}$ also has applications to quantum circuits: this algorithm can be used to decide whether a given set of quantum gates is complete (*complete* means that any orthogonal transformation can be approximated to any desired accuracy by a quantum circuit made up of gates from the set). Much effort has been devoted to the construction of specific complete sets of gates [DBE95, BBC+95], but no general algorithm for deciding whether a given set is complete was known.

Finally, we note that the proof of Theorem 3.2 does not yield any bound on the complexity of problems (i) and (ii). We hope to investigate this question in future work.

**Acknowledgment.** P.K. would like to thank Etienne Ghys for pointing out reference [OV90]. We are also grateful to the anonymous referee for his very careful reading of the manuscript.

## REFERENCES

[AI99] M. Amano and K. Iwama, *Undecidability on quantum finite automata*, in Proceedings of the 31st ACM Symposium on Theory of Computing, ACM, New York, 1999, pp. 368–375.

[BBC+95] A. Barenco, C. H. Bennett, R. Cleve, D. P. DiVincenzo, N. H. Margolus, P. W. Shor, T. Sleator, J. A. Smolin, and H. Weinfurter, *Elementary gates for quantum computation*, Phys. Rev. A, 52 (1995), pp. 3457–3467.

[BPR96] S. Basu, R. Pollack, and M.-F. Roy, *On the combinatorial and algebraic complexity of quantifier elimination*, J. ACM, 43 (1996), pp. 1002–1045.

[BC03] V. D. Blondel and V. Canterini, *Undecidable problems for probabilistic automata of fixed dimension*, Theory Comput. Syst., 36 (2003), pp. 231–245.

[Ber75] A. Bertoni, *The solution of problems relative to probabilistic automata in the frame of the formal languages theory*, in Vierte Jahrestagung der Gesellschaft für Informatik, Lecture Notes in Comput. Sci. 26, Springer, Berlin, 1975, pp. 107–112.

[BMT77] A. Bertoni, G. Mauri, and M. Torelli, *Some recursively unsolvable problems relating to isolated cutpoints in probabilistic automata*, in Proceedings of the 4th International

Colloquium on Automata, Languages and Programming, Lecture Notes in Comput. Sci. 52, Springer, Berlin, 1977, pp. 87–94.

[BP02] A. Brodsky and N. Pippenger, *Characterizations of* 1*-way quantum finite automata*, SIAM J. Comput., 31 (2002), pp. 1456–1478.

[BT97] V. D. Blondel and J. N. Tsitsiklis, *When is a pair of matrices mortal?*, Inform. Process. Lett., 63 (1997), pp. 283–286.

[CLO92] D. Cox, J. Little, and D. O'Shea, *Ideals, Varieties, and Algorithms*, Undergrad. Texts Math., Springer, New York, 1992.

[DBE95] D. Deutsch, A. Barenco, and A. K. Ekert, *Universality in quantum computation*, Proc. Roy. Soc. London Ser. A, 449 (1995), pp. 669–677.

[DJK03] H. Derksen, E. Jeandel, and P. Koiran, *Quantum automata and algebraic groups*, J. Symbolic Comput., 39 (2005), pp. 357–371.

[Jea02] E. Jeandel, *Indécidabilité sur les automates quantiques*, Master's Thesis, 2002; available online from http://perso.ens-lyon.fr/emmanuel.jeandel/publis.html.

[KW97] A. Kondacs and J. Watrous, *On the power of quantum finite state automata*, in Proceedings of the 38th Annual Symposium on Foundations of Computer Science, IEEE, Los Alamitos, 1997, pp. 66–75.

[MC00] C. Moore and J. Crutchfield, *Quantum automata and quantum grammars*, Theoret. Comput. Sci., 237 (2000), pp. 257–306.

[MS05] Y. Matiyasevich and G. Sénizergues, *Decision problems for semi-Thue systems with a few rules*, Theoret. Comput. Sci., 330 (2005), pp. 145–169.

[OV90] A. Onishchik and E. Vinberg, *Lie Groups and Algebraic Groups*, Springer, Berlin, 1990.

[Paz71] A. Paz, *Introduction to Probabilistic Automata*, Academic Press, New York, 1971.

[Pos46] E. L. Post, *A variant of a recursively unsolvable problem*, Bull. Amer. Math. Soc., 52 (1946), pp. 264–268.

[Rab63] M. O. Rabin, *Probabilistic automata*, Inform. Control, 6 (1963), pp. 230–245.

[Rab67] M. O. Rabin, *Mathematical theory of automata*, in Proc. Sympos. Appl. Math. 19, AMS, Providence, RI, 1967, pp. 153–175.

[Ren92a] J. Renegar, *On the computational complexity and geometry of the first-order theory of the reals.* I, J. Symbolic Comput., 13 (1992), pp. 255–299.

[Ren92b] J. Renegar, *On the computational complexity and geometry of the first-order theory of the reals.* II, J. Symbolic Comput., 13 (1992), pp. 301–327.

[Ren92c] J. Renegar, *On the computational complexity and geometry of the first-order theory of the reals.* III, J. Symbolic Comput., 13 (1992), pp. 329–352.

[Su90] F. E. Su, *The Banach-Tarski Paradox*, Minor Thesis, 1990.

[Sw58] S. Swierczkowski, *On a free group of rotations of the Euclidean space*, Nederl. Akad. Wetensch. Proc. Ser. A, 20 (1958), pp. 376–378.

[Sw94] S. Swierczkowski, *A class of free rotation groups*, Indag. Math. (N.S.), 5 (1994), pp. 221–226.

SIAM J. COMPUT.
Vol. 34, No. 6, pp. 1474–1492

# NOVEL TRANSFORMATION TECHNIQUES USING Q-HEAPS WITH APPLICATIONS TO COMPUTATIONAL GEOMETRY[*]

QINGMIN SHI[†] AND JOSEPH JAJA[†]

**Abstract.** Using the notions of Q-heaps and fusion trees developed by Fredman and Willard, we develop general transformation techniques to reduce a number of computational geometry problems to their special versions in partially ranked spaces. In particular, we develop a fast fractional cascading technique, which uses linear space and enables sublogarithmic iterative search on catalog trees in the case when the degree of each node is bounded by $O(\log^\epsilon n)$ for some constant $\epsilon > 0$, where $n$ is the total size of all the lists stored in the tree. We apply the fast fractional cascading technique in combination with the other techniques to derive the first linear-space sublogarithmic time algorithms for two fundamental geometric retrieval problems: orthogonal segment intersection and rectangular point enclosure.

**Key words.** searching, computational geometry, geometric retrieval, fractional cascading, orthogonal segment intersection, rectangular point enclosure

**AMS subject classifications.** 68P10, 68P05, 68Q25

**DOI.** 10.1137/S0097539703435728

**1. Introduction.** Q-heaps and fusion trees [10, 11] are data structures that achieve sublogarithmic search time on one-dimensional (1-D) data. In particular, a Q-heap supports constant time *insertion*, *deletion,* and *predecessor* search on very "small" subsets of a larger set using linear space. In [30], Willard illustrated how upper bounds for several search problems can be improved using the Q-heap. In this paper, we further explore the Q-heap technique in the context of computational geometry by using it to develop several general techniques, which lead to faster algorithms for a number of geometric retrieval problems.

A geometric retrieval problem is to preprocess a set $S$ of $n$ geometric objects so that, when given a query specifying a set of geometric constraints, the subset $Q$ of $S$ consisting of the objects that satisfy these constraints can be reported quickly. Examples of geometric retrieval problems include orthogonal segment intersection [26, 3], rectangular point enclosure [19, 27, 3], and orthogonal range queries [2, 29, 20, 3, 4, 21, 25] and their special cases [19, 30, 5, 18]. A typical data structure for handling such a problem often involves a primary constant-degree search tree $T$ whose nodes are each equipped with secondary structures, which are built on a subset of $S$ and are capable of handling special versions of the original query very quickly. There are two main ideas behind such a typical data structure. First, the objects in $S$ are distributed among the nodes of $T$ in such a way that the number of nodes of $T$ visited during a search process is bounded by the depth of $T$ or the output size. Second, for each node $v$ visited, the search query on the set $S(v)$ of objects stored there can be performed very quickly. The first idea is often realized by ensuring that either a nonroot node $v$ is visited only if it is on a specific path from the root of $T$

[*]Received by the editors October 1, 2003; accepted for publication (in revised form) March 15, 2005; published electronically August 17, 2005. This research was supported in part by the National Science Foundation through the National Partnership for Advanced Computational Infrastructure (NPACI), DoD-MD Procurement under contract MDA90402C0428, and by NASA under the ESIP Program NCC5300.

http://www.siam.org/journals/sicomp/34-6/43572.html

[†]Institute for Advanced Computer Studies, Department of Electrical and Computer Engineering, University of Maryland, College Park, MD 20742 (qshi@umiacs.umd.edu, joseph@umiacs.umd.edu).

to a leaf node (for example, in handling the segment intersection and rectangle point enclosure problems [3]), or the time spent at $v$ can be compensated by the time spent at reporting $Q \cap S(parent(v))$, in case searching $S(v)$ is not fruitful; i.e., we visit $v$ only if $S(parent(v)) \cap Q \neq \emptyset$ (for example, in handling the three-sided three-dimensional (3-D) range queries [19] and the 3-D dominance queries [5]). The second idea is often implemented by (i) making sure that we can perform certain types of *rank* operations on $S(v)$ in constant time (typically using the fractional cascading technique), or (ii) constructing a constant number of secondary data structures on $S(v)$ such that each can be chosen to handle a special version of the original query, which is derived based on the additional information provided by the discriminator associated with $v$ and which has less constraints.

The fusion tree technique makes it possible to further reduce the search complexity of some of these structures by increasing the degree of the primary search trees to $\log^\epsilon n$, so that the height of the tree is reduced to $O(\log n / \log\log n)$. The *branch* operation at each node can be performed in constant time by making direct use of the Q-heaps. However, the fact that the degree of the tree is now dependent on $n$ introduces at least the following three problems. First, the standard fractional cascading technique would take a nonconstant amount of time at each node. In fact, a search operation performed on the list associated with a tree node would require $O(\log\log n)$ time, which negates the effect of a “fattened” primary search tree. Second, the number of discriminators at a node $v$ is no longer a constant, which makes it very difficult to use only a constant number of secondary structures for the quick handling of special versions of the original query. Finally, a nonempty output at a node no longer ensures that we can afford to search each of its children, since none of them may contain an output object. Doing so would result in an $O(f \log^\epsilon n)$ term in the overall search complexity, where $f$ is the output size.

In this paper, we develop several techniques to handle the first two problems by making strong use of the Q-heap technique. In particular, we show that, under the RAM model used by Fredman and Willard, if $T$ is a tree whose degree $c$ is bounded by a logarithmic function of $n$, i.e., $c = \log^\epsilon n$ for some constant $\epsilon$, then it is possible to improve the performance of the standard fractional cascading structure so that *rank* operations on $S(v)$ for each node $v$ except the root can be performed in constant time (independent of $n$) without asymptotically increasing the storage cost. We call this new fractional cascading technique *fast fractional cascading*. This is a general technique and is of independent interest. We also show how to perform search operations on a set $S(v)$ very quickly by transforming the problem to the partially ranked space $[1, 2, \ldots, c] \times \mathcal{N}$, where very fast search operations are possible using Q-heaps and table look-ups. This transformation can also be used to handle the so-called *lookahead* problem [30], deciding whether or not searching a substructure will be fruitful before it is actually searched, which is crucial in overcoming the third obstacle mentioned at the end of the last paragraph.

We believe that these techniques have the potential to improve the asymptotic upper bounds of a wide range of geometrical retrieval problems. In this paper we apply our techniques to two fundamental geometric retrieval problems: *orthogonal segment intersection* and *rectangular point enclosure*. Each of our algorithms achieves $O(n)$ space and $O(\log n / \log\log n + f)$ query time. The best previous results require $O(\log n + f)$ query time when using linear space [3, 18]. In a separate paper [14], we show how to apply these techniques to obtain the first linear-space sublogarithmic algorithm for the 3-D dominance reporting problem.

We now formally define the two geometric retrieval problems to be tackled in this paper. To facilitate our explanation, we will denote a horizontal (resp., vertical) segment in a 2-D space as $(x_1, x_2; y)$ (resp., $(x; y_1, y_2)$), where $(x_1, y)$ and $(x_2, y)$ (resp., $(x, y_1)$ and $(x, y_2)$) are its two endpoints and $x_1 \leq x_2$ (resp., $y_1 \leq y_2$)).

1. *Orthogonal segment intersection.* Given a set $S$ of $n$ horizontal segments, report the subset $Q$ of segments that intersect a given vertical segment. We say a horizontal segment $(x_1, x_2; y)$ *intersects* a vertical segment $(x; y_1, y_2)$ if and only if $x_1 \leq x \leq x_2$ and $y_1 \leq y \leq y_2$. We call the segments in $Q$ *proper segments* relative to the given query.

2. *Rectangular point enclosure.* Let $(x_1, x_2; y_1, y_2)$ denote a rectangle in a 2-D space with edges parallel to the axes, where the intervals $[x_1, x_2]$ and $[y_1, y_2]$ are the projections of this rectangle to the $x$-axis and $y$-axis, respectively. Given a set $S$ of rectangles, report the subset $Q$ of *proper rectangles* such that each rectangle $(x_1, x_2; y_1, y_2)$ in $Q$ contains a query point $(x, y)$, i.e., $x_1 \leq x \leq x_2$ and $y_1 \leq y \leq y_2$.

In this paper, we use the RAM model as described in [10]. In this model, it is assumed that each word contains $w$ bits, and the size of a data set never exceeds $2^w$, i.e., $w \geq \log_2 n$. In addition to arithmetic operations, bitwise logical operations are also assumed to take constant time.

The next section introduces some well-known techniques that will be heavily utilized in the rest of the paper. In section 3, we present the fast fractional cascading structure, while sections 4 and 5 present the improved algorithms for, respectively, orthogonal segment intersection and rectangular enclosure.

## 2. Preliminaries.

Given a set $S$ of multidimensional points $(x_1, x_2, \ldots, x_d)$, a point with the largest $x_i$-coordinate smaller than or equal to a real number $\alpha$ is called the $x_i$-*predecessor* of $\alpha$, and the one with the smallest $x_i$-coordinate larger than or equal to $\alpha$ is called the $x_i$-*successor* of $\alpha$.

The notion of a *Cartesian* tree was first introduced by Vuillemin [28] (and rediscovered by Seidel and Aragon [23]). A Cartesian tree is a binary tree defined over a finite set of 2-D points sorted by their $x$-coordinates, say $(p_1, \ldots, p_n)$. Let $p_i$ be the point with the largest $y$-coordinate. Then $p_i$ is associated with the root $w$ of $C$. The two children are, respectively, the root of the Cartesian trees built on $p_1, \ldots, p_{i-1}$ and $p_{i+1}, \ldots, p_n$. Note that the left (resp., right) child of $w$ does not exist if $i = 1$ (resp., $i = n$). Figure 2.1 shows an example of the Cartesian tree.

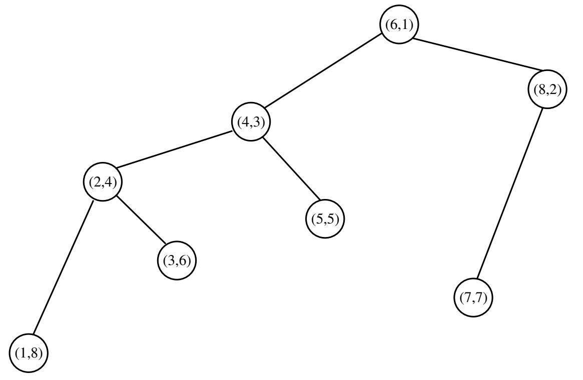


FIG. 2.1. *Cartesian tree.*

**2.1. Cartesian trees.** An important property of the Cartesian tree is given by the following observation [12].

OBSERVATION 2.1. *Consider a set $S$ of 2-D points and the corresponding $(x, y)$-Cartesian tree $C$. Let $x_1 \leq x_2$ be the x-coordinates of two points in $S$, and let $\alpha$ and $\beta$ be their respective nodes in $C$. Then the point with the largest y-coordinate among those points whose x-coordinates are between $x_1$ and $x_2$ is stored in the nearest common ancestor of $\alpha$ and $\beta$.*

Using Observation 2.1, combined with the techniques for computing the nearest common ancestors [13] (see also [1]) in constant time, we have shown in [24] that we can handle the so-called *three-sided* 2-D *range queries* efficiently. Briefly, a point $(a, b)$ satisfies the three-sided query $(x_1, x_2, y)$, with $x_1 \leq x_2$, if $x_1 \leq a \leq x_2$ and $b \geq y$.

LEMMA 2.1. *By preprocessing a set of $n$ 2-D points to construct an $(x, y)$-Cartesian tree $C$, we can handle any three-sided 2-D range query given as $(x_1, x_2, y)$, with $x_1 \leq x_2$, in $O(t(n) + f)$ time, where $t(n)$ is the time it takes to find in $C$ the leftmost and rightmost nodes whose x-coordinates fall within the range $[x_1, x_2]$, and $f$ is the number of points reported.*

Note that $C$ should be transformed into a suitable form to enable the computation of nearest common ancestors in constant time.

**2.2. Q-heaps and fusion trees.** *Q-heaps* and *fusion trees*, developed by Fredman and Willard [10, 11], achieve sublogarithmic search time on 1-D data. While the Q-heap data structure was proposed later than the fusion tree, it can be used as a building block for the fusion tree [30]. Using Q-heaps and fusion trees, Willard demonstrated in [30] theoretical improvements for a number of range search problems.

The Q-heap [11] supports insert, delete, and search operations in constant time for small subsets of a large data set of size $n$. Its main properties are given in the following lemma (the version presented here is taken from [30]).

LEMMA 2.2. *Suppose $S$ is a subset with cardinality $m < \log^{1/5} n$ lying in a larger database consisting of $n$ elements. Then there exists a Q-heap data structure of size $O(m)$ that enables insertion, deletion, member, and predecessor queries on $S$ to run in constant worst-case time, provided that access is available to a precomputed table of size $o(n)$.*

Note that the look-up table of size $o(n)$ referred to above is shared by all the Q-heaps built on subsets of this larger database.

The fusion tree built on the Q-heap achieves linear space and sublogarithmic search time. The following lemma is a simplified version of Corollary 3.2 from [30].

LEMMA 2.3. *Assume that in a database of $n$ elements we have available the use of precomputed tables of size $o(n)$. Then it is possible to construct a data structure of size $O(n)$ space, which has a worst-case time $O(\log n / \log \log n)$ for performing member, predecessor, and rank operations.*

Notice that the assumptions of the RAM model introduced in section 1 are critical to achieving the bounds claimed in the above lemmas.

**2.3. Adjacency map and hive graph.** The notion of the *adjacency map* was first introduced by Lipski and Preparata [16]. Given a set $S$ of $n$ horizontal segments, the *vertical adjacency map* $G(S)$ is constructed by interconnecting the horizontal segments in $S$ using vertical (infinite, semi-infinite, or finite) segments as follows: from each endpoint of the segments in $S$, draw two rays shooting upward and downward, respectively, until they meet other segments in $S$ except possibly at an endpoint. This creates a planar subdivision $G(S)$ with $O(n)$ vertices, which are the joints of

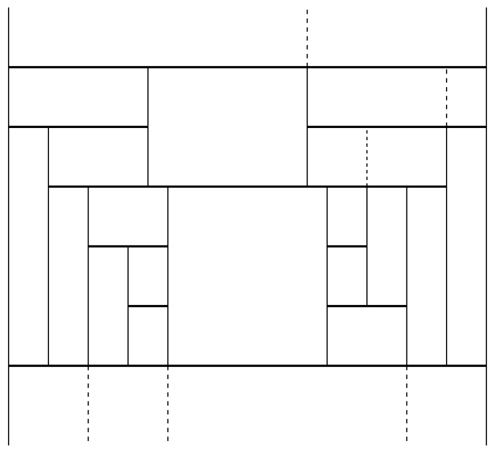


FIG. 2.2. *A hive graph.*

the horizontal and vertical segments. $G(S)$ can be represented in $O(n)$ space using the adjacency lists associated with the vertices. We call the edges supported by the horizontal segments *horizontal* edges and those supported by the vertical segments *vertical* edges.

Chazelle noticed in [3] that the adjacency map is a useful tool which, when modified appropriately, can be used to handle the orthogonal segment intersection problem efficiently. His modification of the vertical adjacency map is called the *hive graph.* A hive graph $H(S)$ is derived from $G(S)$ by adding only vertical segments to $G(S)$ while maintaining $O(n)$ vertices and $O(n)$ space representation. However, it has the important property that each face may have, in addition to its four (or fewer) corners, at most two extra vertices, one on each horizontal edge. Figure 2.2 shows a vertical adjacency map and its corresponding hive graph, in which the additional vertical edges are depicted as dashed lines. By assuming that the endpoints of the segments in $S$ all have distinct $x$- and $y$-coordinates, as [3] did, one can conclude that each face of $H(S)$ has $O(1)$ vertices on its boundary. Given a query segment $(x; y_1, y_2)$, the segment intersection query can be handled as follows. We first find the face in $H(S)$ that contains the endpoint $(x, y_1)$ in $O(\log n)$ time by using one of the well-known planar point location algorithms [8, 9, 15, 17, 22]. Then we traverse a portion of $H(S)$ from bottom up, following the direction from $(x, y_1)$ to $(x, y_2)$. Only a constant number of vertices are visited between two consecutive encounters of the horizontal edges that intersect the query segment.

Note that the vertical boundary of a face of the hive graph corresponding to a vertical adjacency map will not necessarily contain a constant number of vertices if the assumption that the endpoints of the segments in $S$ have distinct $x$-coordinates does not hold. Since we will need to deal later with such a case, we get around this problem by associating with each vertex pointers to the upper-right or upper-left corner in the same face, as follows. We modify $H(S)$ by associating with each vertex $\beta$ two additional pointers $p(\beta)$ and $q(\beta)$. Let $\beta = \delta_1, \delta_2, \ldots, \delta_l = \gamma$ be the maximal chain of vertices such that each pair of consecutive vertices $\delta_i$ and $\delta_{i+1}$ is connected by a vertical edge $e_i$ and $\delta_{i+1}$ is above $\delta_i$, for $i = 1, \ldots, l-1$. Note that this chain can be empty ($l = 1$), in which case both $p(\beta)$ and $q(\beta)$ are null. If there exists a vertex in $\{\delta_2, \delta_3, \ldots, \delta_l\}$ which has a horizontal edge connecting it to a

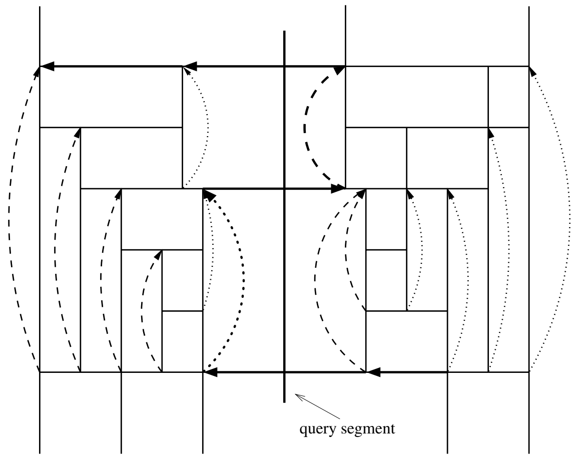


FIG. 2.3. *Modified hive graph.*

vertex to the left of it, then $p(\beta)$ points to the lowest such vertex. Otherwise, $p(\beta)$ is null. Similarly, if there exists a vertex that has a horizontal edge connecting it to a vertex to the right of it, then $q(\beta)$ points to the lowest such vertex. Otherwise, $q(\beta)$ is null. It is easy to see that, using these additional pointers, we can in constant time reach the next proper segment without the distinct $x$-coordinates assumption. Figure 2.3 shows such a modified hive graph. The additional pointers $p(\beta)$ and $q(\beta)$ are depicted, respectively, as dashed and dotted arrows. To simplify the drawing, we omit the pointer $\alpha = p(\beta)$ or $\alpha = q(\beta)$ if $\alpha$ is null or $(\alpha, \beta)$ is an edge in $H(s)$. This figure also illustrates the search path of an exemplary segment intersection query by highlighting the pointers involved.

As noted in [3], when the query segment is semi-infinite, that is, consists of a ray $(x; -\infty, y_2)$ shooting downward, there is no need to perform the initial planar point location query. Instead, we can, during preprocessing, sort the $x$-coordinates of the vertical edges of the faces unbounded from below, and perform as the first step at query time a search on the sorted $x$-coordinates to locate the face that contains the point $(x, -\infty)$. The following lemma is a restatement of Corollary 1 in [3].

LEMMA 2.4. *Given a set $S$ of $n$ horizontal segments in the plane, an $O(n)$ space hive graph can be used to determine all the intersections of the horizontal segments with a semi-infinite vertical query segment $s = (x; -\infty, y_2)$ in $O(t(n)+f)$ time, where $f$ is the number of intersections, and $t(n)$ is the time it takes to search the sorted list of $x$-coordinates.*

Combining Lemmas 2.3 and 2.4, we have the following corollary.

COROLLARY 2.5. *Given a set $S$ of $n$ horizontal segments in the plane and a vertical query segment in the form of $(x; -\infty, y_2)$, it is possible to report all $f$ proper segments of $S$ in $O(\log n/\log\log n + f)$ time using $O(n)$ space.*

Clearly, by rotating the hive graph 90° clockwise (resp., counterclockwise), the same type of techniques will yield a solution for handling any orthogonal segment intersection query that involves a set $S$ of vertical segments and a horizontal query segment of the form $(-\infty, x_2; y)$ (resp., $(x_1, +\infty; y)$). We will denote this rotated hive graph HL(S) (resp., $HR(S)$).

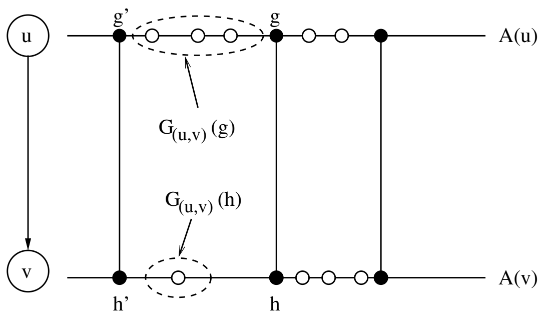


FIG. 3.1. *Fractional cascading.*

**3. Fast fractional cascading.** Suppose we have a tree $T = (V, E)$ rooted at $w$ such that each node $v$ has a degree bounded by $c$ and contains a catalog $L(v)$ of sorted elements. Let $n$ denote the total number of elements in these catalogs. A key value $k(g)$ from $\mathcal{N} \cup \{-\infty, +\infty\}$ is associated with each element $g$ in $L(v)$. The elements in $L(v)$ do not need to have distinct key values. We call such a tree a *catalog tree.* Let $x$ be a real number and $F$ be an arbitrary forest with $p$ nodes consisting of subtrees of $T$ determined by some of the children of $w$. Both $x$ and $F$ can be specified online, i.e., not necessarily at preprocessing time. Let $\sigma_L(x)$ denote the successor of $x$ in a catalog $L$. The *iterative search problem* is defined as follows [6]: report $\sigma_{L(v)}(x)$ for each $v$ in $F$. This problem (in a somewhat less general form) was first discussed by Willard in the context of handling 2-D orthogonal range queries [29]. A technique called *fractional cascading* was later proposed by Chazelle and Guibas [6, 7] to deal with the general problem. We now briefly introduce their approach.

**3.1. Fractional cascading.** The following lemma is a direct derivation from the one given by Chazelle and Guibas for identifying the successor of a value $x$ in each of the catalogs in $F$ [6].

LEMMA 3.1. *There exists a linear size fractional cascading data structure that can be used to determine the successors of a given value $x$ in the catalogs associated with $F$ in $O(p \log c + t(n))$ time, where $t(n)$ is the time it takes to identify the successor of $x$ in $L(w)$.*

The main component of a fractional cascading structure is the notion of the *augmented catalogs.* At each node $v$ in $T$, in addition to the original catalog $L(v)$, we store another augmented catalog $A(v)$, which is a superset of $L(v)$ and contains additional copies of elements from the augmented lists associated with its parent and children. With each element $h$ in $A(v)$, we associate a pointer to its successor $\sigma_{L(v)}(h)$ in $L(v)$. Since $A(v)$ is a superset of $L(v)$, we have $\sigma_{L(v)}(g) = \sigma_{L(v)}(\sigma_{A(v)}(g))$. Note that the elements in an augmented list $A(v)$ form a multiset $S(v)$; that is, a single element can appear multiple times in an augmented list. The elements in an augmented list are chained together to form a doubly linked list.

As illustrated in Figure 3.1, let $u$ and $v$ be two neighboring nodes in $T$, $u$ being $v$'s parent. There exists a subset $B(u, v)$ of $A(u) \times A(v)$ such that, for each pair of elements $(g, h) \in B(u, v)$, $k(g) = k(h)$. The pair of elements $(g, h)$ are called a *bridge.* There is a pointer to $h$ associated with the element $g$, and similarly a pointer to $g$ is associated with $h$. We will call $g$ a *down-bridge*, and $h$ an *up-bridge*, associated with the edge $(u, v)$. It is important to point out that each element in an augmented list can serve as at most one up-bridge or one down-bridge, but not both. Bridges respect the ordering of equal-valued elements and thus do not "cross." This guarantees that $B(u, v)$ can be ordered and the concept of *gap* presented next is well defined. In this ordered set $B(u, v)$, the bridge $(g, h)$ appears after the $(g', h')$ if and only if $g$ appears after $g'$ in $A(u)$. A *gap* $G_{(u,v)}(g, h)$ of bridge $(g, h)$ is defined as the multiset of elements from both $A(u)$ and $A(v)$ which are strictly between two bridges $(g, h)$ and $(g', h')$, where $(g', h')$ is the bridge that appears immediately before $(g, h)$ in $B(u, v)$. Accordingly, we define the *up-gap* (resp., *down-gap*) $G_{(u,v)}(g)$ (resp., $G_{(u,v)}(h)$) as the subset of $G_{(u,v)}(g, h)$ containing elements from $A(u)$ (resp., $A(v)$), preserving their orders in the respective augmented catalogs.

The fractional cascading structure maintains the invariant that the size of any gap cannot exceed $6c - 1$. Chazelle and Guibas provided in [6] an algorithm that can construct such a data structure in $O(n)$ time, and they prove that it requires $O(n)$ space.

Given a parent-child pair $(u, v) \in E$, suppose we know the successor $\sigma_{A(u)}(x)$ of a value $x$ in $A(u)$. We follow $A(u)$ along the direction of increasing values to the next down-bridge $g$ connecting $u$ and $v$ (it could be $\sigma_{A(u)}(x)$ itself if it is a down-bridge), cross it to its corresponding up-bridge $h$, and scan $A(v)$ in the opposite position until the successor of $x$ in $A(v)$ is encountered. Clearly, $\sigma_{A(v)}(x)$ is guaranteed to be found by this process. The constraint on the gap size ensures that the number of comparisons required is $O(c)$.

When $c$ is a constant, the above result is optimal. When $c$ is large, Chazelle and Guibas used the so-called *star tree* to achieve $O(\log c)$ search time on each catalog except the one stored at the root.

**3.2. Fast fractional cascading.** The fractional cascading structure described above is strictly list-based, and hence all the related algorithms can run on a pointer machine within the complexity bounds stated. Using the variation of the RAM model introduced in section 1 and the Q-heap technique of Fredman and Willard [11], summarized in section 2.2, we can achieve constant search time (independent of $c$) per node for the class of catalog trees whose degree is bounded by $c = \log^{\epsilon} n$, while simultaneously maintaining a linear size data structure. We call this version *fast fractional cascading*. This result improves over our previous result in [24], which achieves the same search complexity but requires nonlinear space. We will first revisit the nonlinear space solution and then explain how to reduce the storage cost to linear.

**3.2.1. Fast fractional cascading with nonlinear space.** We augment the fractional cascading structure described in section 3.1 by adding two types of components to each augmented catalog $A(v)$. First, we associate $c$ additional pointers $p_1(g), p_2(g), \ldots, p_c(g)$ with each element $g$ in $A(v)$ such that $p_i(g)$ points to the next down-bridge (possibly $g$ itself) connecting $v$ to $w_i$, where $w_i$ is the $i$th child of $v$ from the left. Second, we build for each up-gap $G_{(u,v)}(h)$ a Q-heap $Q(h)$, containing elements in $G_{(u,v)}(h)$ with distinct values (choosing the first one if multiple elements have the same value). For large enough $n$ we have $6c - 1 < \log^{1/5} n$, and therefore Lemma 2.2 is applicable. We have added $c$ pointers for each of the elements in the augmented catalogs, whose overall size cannot exceed $O(n)$. In addition, a global look-up table of size $O(n)$ is used to serve all the Q-heaps. Finally, no two up-gaps in an augmented catalog overlap, as they correspond to the same edge in $T$ (which is not true for a general graph). Hence the Q-heaps cannot consume more than $O(n)$ space.

Now suppose we have found $g = \sigma_{A(u)}(x)$ in $A(u)$. Let $v$ be the $i$th child of $u$.

By following the pointer $p_i(g)$, we can reach in constant time the next down-bridge in $u$ and then its companion up-bridge $h$ in $v$. Using $Q(h)$, we can find the successor of $x$ in $G_{(u,v)}(h)$ in constant time.

LEMMA 3.2. *Let $c = O(\log^\epsilon n)$. The fast fractional cascading structure described above allows the identification of the successors of a given value $x$ in the catalogs associated with $F$ in $O(p + t(n))$ time, where $t(n)$ is the time it takes to identify the successor of $x$ in $L(w)$. This structure requires $O(cn)$ space.*

**3.2.2. Fast fractional cascading with linear space.** We partition each augmented catalog $A(u)$ into $p = \lceil |A(u)|/c \rceil$ blocks $B_1, B_2, \ldots, B_p$, each, except possibly the last one, containing $c$ elements. For each block $B_i$ starting from the $l$th element of $A(u)$, we construct a set $C_i$ of $t \le 7c - 1$ records as follows. For each down-bridge $g$ that is the $d$th element in $A(u)$, where $l \le d \le l + 7c - 2$, we include in $C_i$ a record $r$ that contains two entries *r.ptr* and *r.key*. The entry *r.ptr* is a pointer to $g$, and *r.key* is the key of $r$ whose value is defined as $r.key = j * (7c - 1) + (d - l)$ if $g$ is associated with the edge connecting $u$ and its $(j + 1)$th child (note that *r.key* can fit in a word). The records in $C_i$ are sorted in increasing order by their key values. Now let $g$ be the successor of a value $x$ in $A(u)$ and suppose we want to find the successor of $x$ in the augmented catalog associated with the $(j + 1)$th child $v$ of $u$. It is easy to determine in constant time the block $B_i$ to which $g$ belongs and its position $f$ relative to the starting position of $B_i$ ($f = 0$ if $g$ is the first element in $B_i$). If $g$ is itself a down-bridge associated with $(u, v)$, then we are done. Otherwise, due to the invariant regarding the gap size, the next down-bridge $h$ associated with $(u, v)$ must have a corresponding record in $C_i$. The following lemma transforms the problem of finding $h$ to a successor search in $C_i$.

LEMMA 3.3. *The record in $C_i$ that corresponds to $h$ is the successor of the value $y = j \cdot (7c - 1) + f$.*

*Proof.* First we notice the fact that all the keys of the records in $C_i$ are distinct. Let $y' = j \cdot (7c - 1) + f'$ be the key of the record in $C_i$ that corresponds to $h$. It is obvious that $y < y'$. Now let $y'' = j'' \cdot (7c + 1) + f''$ be the key of a record $r$ in $C_i$ such that $y \le y''$. We need to show only that $y' \le y''$. Since both $f''$ and $f$ are nonnegative integers less than $7c - 1$, the fact that $y \le y''$ leads to either $j < j''$, or $j = j''$ and $f \le f''$. If $j < j''$, we immediately have $y' < y''$. On the other hand, if $j = j''$, then the record $r$ also corresponds to a down-bridge associated with the edge $(u, v)$. Since $h$ is the leftmost down-bridge closest to $g$, we have $f' \le f''$. Thus $y' \le y''$. ☐

The problem of finding the successor of an integer value in a small set $C_i$ can be solved, again using the Q-heap data structure. The following straightforward observations ensure the applicability of Lemma 2.2:

1. $|C_i| < \log^{1/5} n$ for $n$ large enough; and
2. the total number of distinct keys created for all the augmented catalogs is bounded by $O(n)$.

Finally, it is easy to see that the overall additional space introduced by the new Q-heaps is $O(n)$, and thus we have the following theorem.

THEOREM 3.4. *For $c = O(\log^\epsilon n)$ for some $\epsilon < \frac{1}{5}$, our fast fractional cascading structure allows the identification of the successors of a given value $x$ in the catalogs associated with $F$ in $O(p + t(n))$ time, where $t(n)$ is the time it takes to identify the successor of $x$ in $L(w)$. This structure requires $O(n)$ space.*

**4. Orthogonal segment intersection.** Before tackling the general orthogonal segment intersection problem, we develop a linear size data structure to handle a special case in which the $x$-coordinates of the endpoints of the segments and the query segment can take integer values only over a small range of values. We will later show how to use the solution of the special case to derive a solution to the general problem.

**4.1. Modified vertical adjacency map.** Assume that the $x$-coordinates of the endpoints of each segment $(k_1, k_2; y)$ in the given set $R$ of $n$ horizontal segments can take values from the set of integers $\{1, 2, \ldots, c\}$, where $c = \log^{\epsilon} n$ is an integer, and furthermore, assume that the $x$-coordinate $k$ of the query segment $r = (k; z_1, z_2)$ is an integer between 1 and $c$. Let $Y(R) = (y_1(R), y_2(R), \ldots, y_{n'}(R))$ be the list of distinct $y$-coordinates of the segments in $R$ sorted in increasing order.

Our overall strategy consists of augmenting the vertical adjacency map with auxiliary structures so that we will be able to identify the lowest segment in $R$ intersecting $r$ very quickly, followed by progressively determining the next sequence of lowest segments, each in $O(1)$ time. The details of this strategy are described next.

Our indexing structure $D(R)$ consists of two major components: $H(R)$ and $M(R)$. $H(R)$ is a *directed* vertical adjacency map with auxiliary information attached to it. We define the direction of the horizontal edges to be from right to left, and that of the vertical edges to be from bottom up. Note that we do not require that each face of $H(R)$ have a constant number of vertices on its boundary.

Each vertex of $H(R)$ is naturally associated with a pair of $x$, $y$-coordinates. We call the vertex with an outgoing horizontal edge a *tail*. We augment $H(R)$ with three types of components as follows.

1. For each distinct $y$-coordinate $y_j(R)$ of $Y(R)$, we create a Q-heap $Q_j(R)$ to index the $x$-coordinates of the vertices whose $y$-coordinates are equal to $y_j(R)$.

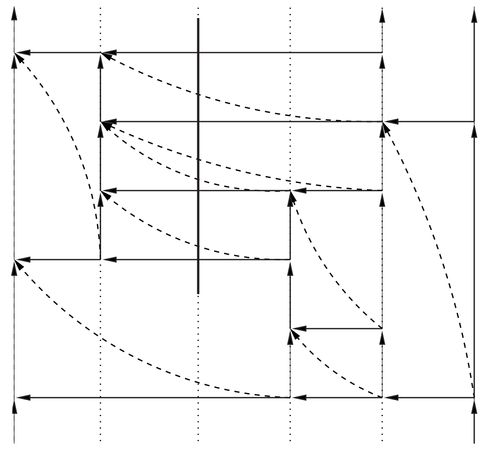
2. For each integer $1 \leq i \leq c$ that serves as the $x$-coordinate of at least one tail, we create a list $P_i(R)$ of records. Each record $g$ corresponds to a tail $\alpha$ whose $x$-coordinate is $i$ and contains two elements: $g.key$, which is the $y$-coordinate of $\alpha$, and $g.ptr$, which is a pointer to $\alpha$. This list is sorted in increasing order by the key values.

3. With each vertex $\beta$ we associate two pointers $p(\beta)$ and $q(\beta)$. Let $y_j(R)$ be the $y$-coordinate of $\beta$. Then $p(\beta)$ points to the Q-heap $Q_j(R)$. If $\beta$ is not a tail, $q(\beta)$ is null. Otherwise, there is at least one vertex with the same $y$-coordinate as $\beta$ that has an outgoing vertical edge and is to the strict left of $\beta$. Let $\gamma$ be the rightmost such vertex, and $e_1, e_2, \ldots, e_l$ be the shortest chain of vertical edges starting from $\gamma$ such that the head $\xi$ of $e_l$ has an incoming horizontal edge. If such a chain exists, then $q(\beta)$ points to $\xi$. If not, $q(\beta)$ is null. Note that intuitively $q(\beta)$ is the top left corner (if it exists) of a face containing $\beta$.

It is clear that $H(R)$ is of size $O(n)$.

In addition to $H(R)$, we have a bitmap $M(R)$ consisting of a list of bit-vectors. Each vector $V_j(R)$ corresponds to a distinct $y$-coordinate $y_j(R)$ and contains $c$ bits. The $i$th bit, starting from the most significant one, is set to 1 if there is a vertical edge in $H(R)$ passing through the point $(i, y_j(R))$ and 0 otherwise. Each vector can easily fit in a single word, and thus the storage cost of $M(R)$ is $O(n)$. These vectors are aligned with the lower end of the words and are stored in increasing order by the values of the corresponding $y$-coordinates.

As an example, Figures 4.1(a) and (b) illustrate the structures $H(R)$ and $M(R)$. In Figure 4.1(a), the dotted lines depict the $c$ possible $x$-coordinates, the dashed pointers are the $q$-pointers that are not null, and the thick line represents the query segment.



(a)

|  | 6 | 5 | 4 | 3 | 2 | 1 |
|---|---|---|---|---|---|---|
| $V_6(R)$ | 1 | 1 | 0 | 0 | 1 | 1 |
| $V_5(R)$ | 1 | 1 | 0 | 0 | 1 | 1 |
| $V_4(R)$ | 1 | 1 | 0 | 1 | 1 | 1 |
| $V_3(R)$ | 1 | 1 | 0 | 1 | 1 | 1 |
| $V_2(R)$ | 1 | 0 | 0 | 1 | 1 | 1 |
| $V_1(R)$ | 1 | 0 | 0 | 1 | 1 | 1 |

(b)

FIG. 4.1. (a) $H(R)$ *and* (b) $M(R)$.

Given a vertical segment $r = (k; z_1, z_2)$, we first identify the lowest segment that intersects $r$ and then report each of the remaining proper segments in the direction of increasing $y$-coordinates.

Locating the lowest segment that intersects $r$ is performed using $M(R)$. Let $y_j(R)$ be the smallest $y$-coordinate greater than or equal to $z_1$. If no such $y$ exists, then there is no segment in $R$ which intersects $r$. Otherwise, we find the largest value $i \leq k$ such that the $i$th bit in $V_j(R)$ is 1. (This number always exists because the vertical edges whose $x$-coordinates are equal to $c$ form a infinite line, and therefore the lowest bits of all the vectors are set to 1.) This can be accomplished by first masking out the highest $w - k$ bits of $V_j(R)$, $w$ being the number of bits in a word, and then locating its most significant bit. In [10], Fredman and Willard describe how to compute the most significant bit of a word in constant time.

After identifying $i$, we use $P_i(R)$ to determine the record $g$ with the smallest key larger than or equal to $z_1$. We can then immediately obtain the vertex $\alpha$ pointed to by *g.ptr*.

LEMMA 4.1. *Let* $(k_\alpha, y_\alpha)$ *be the coordinates of* $\alpha$. *Then for any segment* $(k_1, k_2; y)$ *in* $R$ *such that* $k_1 \leq k$ *and* $y_\alpha > y \geq y_1$, *we have* $k_2 < k$. *That is, any horizontal segment between* $y_\alpha$ *and* $y$ *which starts to the left of* $r$ *ends before meeting* $r$.

*Proof.* The proof is by contradiction. Suppose $k_2 \geq k$. We then have $k_2 < k_\alpha$, because otherwise the vertical line passing through $\alpha$ would have had at least one vertex $\alpha'$ lying on it with its coordinates $(k_{\alpha'}, y_{\alpha'})$ satisfying $k_{\alpha'} = k_\alpha = i$ and $y_{\alpha'} < y_\alpha$, which contradicts the way we chose $\alpha$. Now consider the vertical line passing through the endpoint $(k_2, y)$. Either it passes through the point $(k_2, y_j(R))$ or intersects a horizontal segment whose left endpoint is to the left of $\alpha$ and whose $y$-coordinate is strictly between $y$ and $y_j(R)$. In the first case, we have a contradiction because there would have been a more significant one-bit than $i$ in $V_j(R)$. In the second case, the right endpoint of that horizontal segment has to be to the strict left of $\alpha$, following the same argument for the segment $(k_1, k_2; y)$. By repeatedly applying this argument, we can show that either there is a one-bit in $V_j(R)$ more significant than the $i$, or there is a record in $P_i(R)$ whose key is smaller than $y_\alpha$ but larger than $y_1$, each leading to a contradiction. ∎

LEMMA 4.2. *If $y_\alpha \le y_2$, then the horizontal segment $t = (k_1, k_2; y)$ on which $\alpha$ lies intersects $r$.*

*Proof.* The only possible scenario in which $t$ does not intersect $r$ is when $k_1 > k$. If this is the case, then there has to be a vertical segment $(k_1; y_1', y_2')$ consisting of several edges in $H$ and passing through the point $(k_1, y)$. This segment cannot cross the horizontal line corresponding to $V_j(R)$ because otherwise there would have been a more significant one-bit than the $i$th in $V_j(R)$. Therefore there has to be a horizontal segment $t' = (k_1', k_2'; y_1')$ with $k_2' > k_1 > k$. Lemma 4.1 implies that $k_1' > k$. Repeating this argument will ultimately lead to a contradiction. $\square$

Lemmas 4.1 and 4.2 show that the horizontal segment $t$ on which $\alpha$ lies is the lowest segment that intersects $r$. Using the Q-heap pointed to by $p(\alpha)$, we can find the vertex $\beta$ with the same $y$-coordinate as $\alpha$ and the smallest $x$-coordinate greater than or equal to $k$. Since $t$ intersects $r$, we are sure that $\beta$ is also on $t$. The following lemma explains how to iteratively find the remaining segments that intersect $r$.

LEMMA 4.3. *Let $t$ be a horizontal segment that intersects $r$ and suppose we know the vertex $\beta$ of $H(R)$ on $t$ with the smallest $x$-coordinate $k_\beta$ larger than or equal to $k$. We can in constant time decide whether there is another segment $t'$ above $t$ that intersects $r$, and furthermore, if there is one, identify in constant time such a $t'$ having the smallest $y$-coordinate larger than that of $t$.*

*Proof.* We first give the algorithm to compute the vertex $\beta'$ on $t'$ with the smallest $x$-coordinate $k_{\beta'}$ larger than or equal to $k$. Consider the following cases.

Case 1: $\beta$ has an outgoing vertical edge $e$ and $k = k_\beta$.

Case 1.1: $e$ is an infinite edge, i.e., $e$ is a ray shooting upwards. Then there are no other segments intersecting $r$.

Case 1.2: The edge $e$ is finite. In this case, the vertex $\beta'$ is the head of $e$, and $t'$ is the horizontal segment on which $\beta'$ lies.

Case 2: $\beta$ does not have an outgoing vertical edge $e$ or $k \ne k_\beta$.

Case 2.1: $q(\beta)$ is null. There are no other segments intersecting $r$.

Case 2.2: $q(\beta)$ is not null. $\beta'$ corresponds to the successor of $k$ in the Q-heap pointed to by $p(q(\beta))$, and $t'$ is the horizontal segment on which $\beta'$ lies.

We now show the correctness of this algorithm. We only discuss Case 2, as the correctness of our algorithm for Case 1 is obvious. First consider the case when $q(\beta)$ is null. Since $\beta$ has to be a tail, the vertical ray starting from $\gamma$ (introduced in the definition of $q(\beta)$) shooting upward does not contain a vertex with an incoming horizontal edge. Hence if there were a horizontal segment above $t$ that intersected $r$, $\gamma$ would not be the rightmost vertex to the left of $\beta$ that has an outgoing vertical edge. Hence no segment above $t$ intersects $r$.

We now consider Case 2.2. In this case, $\gamma$ and the chain starting from it always exist. Let $e_1, e_2, \ldots, e_l$ be the chain of vertical edges used to define $q(\beta)$; $\gamma = \delta_1, \delta_2, \ldots, \delta_{l+1} = \xi$ be the sequence of vertices such that for each $1 \le j \le l$, $e_j = (\delta_j, \delta_{j+1})$; and $(k', y_\gamma)$ and $(k', y_\xi)$ be the respective coordinates of $\gamma$ and $\xi$. We claim that (i) no horizontal segment whose $y$-coordinates are strictly between those of $t$ and $t'$ intersects $r$; (ii) the horizontal segment $t'$ on which $\xi$ lies does intersect $r$; and (iii) the successor $\beta'$ of $k$ in the Q-heap pointed to by $p(q(\beta))$ always exists.

To see why the first claim is true, suppose there is a horizontal segment $(k_1', k_2'; y')$ intersecting $r$ that satisfies $y_\gamma < y < y_\xi$. Then it has to be true that $k' < k_1' \le k$. Since we are discussing Case 2, there has to be another horizontal segment $(k_1'', k_2''; y'')$ such that $k' < k_1'' < k$ and $y_\gamma < y'' < y_\xi$. Following similar arguments as in the proof of Lemma 4.1, we can show that either there exists a vertex on $t$ between $\beta$ and $\gamma$ with an outgoing vertical edge, or there exists a vertex $\xi'$ with an incoming horizontal edge such that its coordinate $(k_{\xi'}, y_{\xi'})$ satisfies $k_{\xi'} = k'$ and $y_\gamma < y_{\xi'} < y_\xi$. Either case leads to a contradiction.

To show that $t'$ indeed intersects $r$, we notice that the right endpoint of the horizontal segment of which the horizontal incoming edge of $\xi$ is a part cannot be to the (strict) left of $s$, because otherwise either there would be a chain of vertical edges closer to $\beta$ than the one we have, or there would be a horizontal segment lying vertically between $t$ and $t'$ that intersects $r$, each leading to a contradiction. This also justifies the last claim (iii), and the proof of the lemma is complete. $\square$

LEMMA 4.4. *Given a set $R$ of $n$ horizontal segments in the plane, whose endpoints can only have $c = \log^\epsilon n$ possible x-coordinates $\{1, 2, \ldots, c\}$, it is possible to report using $O(n)$ space all $f$ proper segments of $R$ which satisfy a query $r = (k; y_1, y_2)$, where $k = 1, 2, \ldots, c$, in $O(t(n) + f)$ time, where $t(n)$ is the time it takes to compute the successor of $y_1$ in $Y(R)$ and $P_i(R)$ for some $i = 1, 2, \ldots, c$.*

Note that we can apply the fusion tree to index the distinct $y$-coordinates using linear space so that $t(n) = O(\log n/ \log\log n)$. In the next section, we will show how to use the algorithm of Lemma 4.4 to solve the general orthogonal segment intersection problem. By applying the fast fractional cascading technique, the time $t(n)$ in Lemma 4.4 can be reduced to $O(1)$ except for the initial search, in which $t(n) = O(\log n/ \log\log n)$.

## 4.2. Handling the general orthogonal segment intersection problem.

In this section we consider the general orthogonal segment intersection problem involving a set $S$ of $n$ horizontal segments. To simplify the presentation, we assume that the endpoints of the segments in $S$ have distinct $x$-coordinates. The primary data structure is a tree $T$ of degree $c = \log^\epsilon n$, built on the endpoints of the $n$ segments sorted in increasing order of the $x$-coordinates. Each leaf node $v$ is associated with $c$ endpoints. Let $x_{\mathrm{l}}$ and $x_{\mathrm{r}}$ be the $x$-coordinates of, respectively, the leftmost endpoints associated with $v$ and the leaf node to its immediate right ($x_{\mathrm{r}} = +\infty$ if $v$ is the rightmost leaf node); then the *x-range* of $v$ is defined as $[x_{\mathrm{l}}, x_{\mathrm{r}})$. For an internal node $u$ with $c$ children $v_0, v_1, \ldots, v_{c-1}$, whose corresponding $x$-ranges are $[x_0, x_1), [x_1, x_2), \ldots, [x_{c-1}, x_c)$, its $x$-range is $[x_0, x_c)$. The set of $c-1$ infinite horizontal lines $b_1(u), b_2(u), \ldots, b_{c-1}(u)$, whose $x$-coordinates are $x_1, x_2, \ldots, x_{c-1}$, respectively, are called the *boundaries* of $u$. When the context is clear, we will use $b_i(u)$ to represent its corresponding $x$-coordinate as well.

The segments in $S$ are distributed among the nodes of $T$ as follows. A horizontal segment is associated with an internal node $u$ if it intersects one of the boundaries of $u$ but none of the boundaries of $u$'s ancestors. A segment is associated with a leaf node $v$ if its endpoints both lie within the $x$-range of $v$.

The set $S(v)$ of segments associated with an internal node $v$ is organized into several secondary data structures, as described below and illustrated in Figure 4.2.

1. The $c-1$ boundaries of each node $v$ are indexed by a Q-heap so that, given an arbitrary value $x$, the leftmost boundary $b_i(v)$ that satisfies $x \leq b_i(v)$ can be identified in constant time.

2. With each boundary $b_i(v)$ with $1 \leq i \leq c-1$, we associate two Cartesian trees $L_i(v)$ and $R_i(v)$. The Cartesian tree $L_i(v)$ contains the endpoints of those segments $(x_1, x_2; y)$ in $S(v)$ which satisfy $b_{i-1}(v) < x_1 \leq b_i(v)$ ($b_0(v) = -\infty$) and $x_2 \geq b_i(v)$, and is used to answer the three-sided range query of the form $(x_1 \leq a, b \leq y \leq d)$; $R_i(v)$ contains the endpoints of those segments that satisfy $b_i(v) \leq x_2 < b_{i+1}(v)$

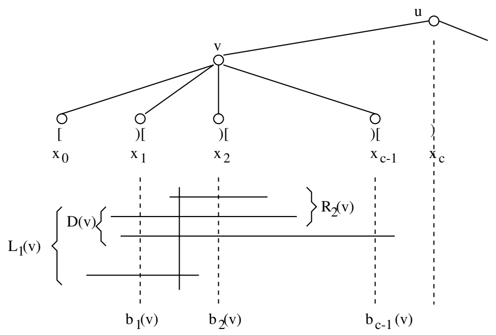


FIG. 4.2. *Data structures for the segments associated with node $v$.*

($b_{i+1}(v) = +\infty$ for $i = c-1$) and $x_1 \le b_i(v)$, and is used to answer the three-sided range query of the form $(x_2 \ge a, b \le y \le d)$. Each Cartesian tree thus created has its nodes doubly linked in order of increasing $y$-coordinates.

3. Let $S'(v)$ be a subset of $S(v)$ containing segments that each intersect at least two boundaries of $v$. We organize these segments using the data structure $D(v)$ discussed in section 4.1. We will later explain how to transform the problem corresponding to $S'(v)$ into the one discussed in section 4.1.

The number of horizontal segments associated with a leaf node is at most $c/2$ since there are only $c$ different endpoints associated with a leaf node, which are simply stored in a list.

We analyze the storage cost of the structures involved in our overall data structure. Obviously, each segment in $S$ is associated with exactly one node $v$ of $T$. For any segment associated with an internal node $v$, it appears in at most three secondary structures, once in $L_i(v)$ associated with the leftmost boundary that $b_i(v)$ it intersects, once in $R_j(v)$ associated with the rightmost boundary $b_j(v)$ that it intersects, and possibly once in $D(v)$. Any segment associated with a leaf node is stored exactly once. Note that all these data structures are linear-space. Hence the total amount of space used by these structures is $O(n)$.

We next outline our search algorithm and then fill in the details as we go along. Let $s = (a; b, d)$ be a vertical segment. To avoid the tedious but not difficult task of treating special cases, we make the assumption that the endpoints of $s$ are different from any of the endpoints of the segments in $S$. To compute the set of proper segments in $Q$, we recursively search the tree $T$, starting from the root. Let $v$ be the node we are currently visiting. We search $v$ as follows.

1. If $v$ is a leaf node, check each segment associated with $v$ and report those that intersect $s$, after which the algorithm terminates.
2. If $x$ lies outside the $x$-range of $v$, then no segment in $S$ intersect $s$, and the algorithm terminates. (This can happen only at the root, when $s$ is to the left of all the segments in $S$.)
3. Otherwise do the following:
   3.1. Find the pair of consecutive boundaries $b_i(v)$ and $b_{i+1}(v)$ of $v$ such that $b_i(v) < a < b_{i+1}(v)$. (The boundary $b_i(v)$ does not exist if $x < b_1(v)$; and $b_{i+1}(v)$ does not exist if $a > b_{c-1}(v)$.)
   3.2. If $b_i(v)$ exist, use $R_i(v)$ to report segments $(x_1, x_2; y)$ that satisfy $x_2 \ge a$ and $b \le y \le d$.
   3.3. If $b_{i+1}(v)$ exist, use $L_{i+1}(v)$ to report those segments $(x_1, x_2; y)$ that

satisfy $x_1 \le a$ and $b \le y \le d$.

3.4. If both $b_i(v)$ and $b_{i+1}(v)$ exist, use $D(v)$ to report those proper segments with no endpoints in the interval $(b_i(v), b_{i+1}(v))$.

3.5. Recursively visit the $(i+1)$th child of $v$ (the first child being the leftmost).

The correctness of the algorithm is obvious, provided that step 3.4 can be performed correctly, a fact we will show shortly. First we note that step 3.1 can be done in constant time using the Q-heap. Furthermore, the access of the Cartesian trees in steps 3.2 and 3.3 can be done in time proportional to the number of segments reported if the successor of $b$ and the predecessor of $d$ in the list of nodes for each Cartesian tree can be identified in constant time. We will show later that we can indeed achieve this goal by applying the fast fractional cascading structure. Finally, it is clear that only one node is visited at each level of $T$, which consists of $O(\log n/\log\log n)$ levels.

Now we focus on step 3.4. The difficulty is to keep the size of $D(v)$ linear and at the same time be able to execute this step in time proportional to the number of segments reported. Let $n'$ be the size of $S'(v)$. One obvious choice is to keep $D(v)$ as $O(c^2)$ lists of segments. Each list corresponds to a pair of boundaries and consists of segments sorted by their $y$-coordinates that cross both boundaries. The storage cost is obviously $O(n')$. However, we will have to visit each list to report the proper segments, since there is no obvious way to decide beforehand which lists contain at least one proper segment (as Willard cleverly did in the design of the fusion priority tree [30]). At least $\Omega(\log^{2\epsilon} n)$ time seems to be required as a result. On the other hand, we can associate with each pair of consecutive boundaries the sorted list of segments that crosses both of them. This approach satisfies the requirement on the query complexity but increases the storage cost by a factor of $\log^{\epsilon} n$.

We now present our solution to handle these segments. We first transform the $x$-coordinates $x_1$ and $x_2$ of the endpoints of each segment $s$ into two integers $k_1$ and $k_2$ between 1 and $c-1$. More specifically, $k_1$ and $k_2$ are the indices of the leftmost and rightmost boundaries of $v$ crossed by $s$. By doing this, we transform $S'(v)$ into another set $W(v)$, in which the segments have their $y$-coordinates unchanged but their $x$-coordinates replaced by the indices of the boundaries. At query time, we also transform the query segment $s = (x; y_1, y_2)$ into another segment $r$ by replacing its $x$-coordinate with the index $k$ of the boundary to its immediate right. It is straightforward to see that a segment in $S'(v)$ is proper if and only if its corresponding segment $(k_1, k_2; y)$ in $W(v)$ satisfies $k_1 < k \le k_2$ and $y_1 \le y \le y_2$. (In the case where $k_1 = k$, the original segment corresponding to $(k_1, k_2; y)$ is already found using $L_{k_1}(v)$ and thus need not be reported here.) We now have exactly the problem we tackled in section 4.1. Hence by Lemma 4.4, we can find the $f'$ proper segments in $W(v)$ in $O(f')$ time, provided that we can in constant time identify the successor of $b$ in the various sorted lists of $y$-coordinates associated with $H(v)$.

To complete the description of our algorithm, we show how to apply the fast fractional cascading structure to search the sorted lists at different levels of the tree. The sorted lists stored at each node $v$ consist of the $2(c-1)$ lists $R_i(v)$ and $L_i(v)$ for $i = 1, \ldots, c-1$; the list of vectors in $M(v)$; and up to $c-1$ lists of $P_i(v)$. Note that during the query time, we need to search only $O(1)$ such lists at each level. Using the fusion tree, we can search the relevant list at the root of $T$ in $O(\log n/\log\log n)$ time.

To see how the various lists are linked through fast fractional cascading, we can imagine a virtual forest $F$ consisting of $c$ "virtual" trees $T_1, T_2, \ldots, T_c$ of degree $3c-2$, such that the lists stored at the roots are $L_1(v), L_2(v), \ldots, L_{c-1}(v), R_{c-1}(v)$, respectively, where $v$ is the root of our search tree. The children of the root storing $L_i(v)$ contain the lists of the $i$th children of $v$ from the left; and the children of the root

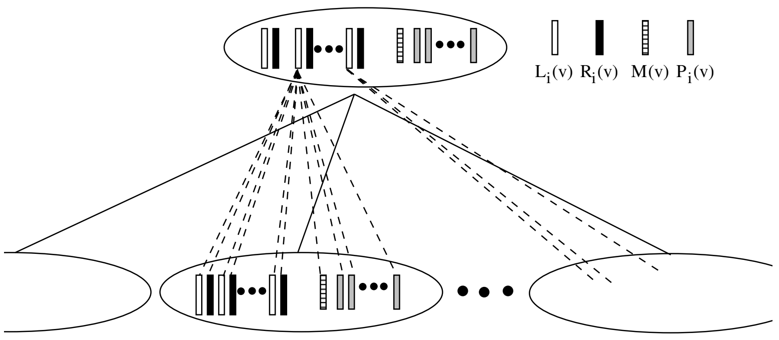


FIG. 4.3. *The virtual forest.*

storing $R_{c-1}(v)$ contain the lists of the $c$th children of $v$. Figure 4.3 illustrates the concept of the virtual forest. It is straightforward to see that a node in $F$ is searched only if its parent is searched. Since $c = \log^\epsilon n$ with $\epsilon < 1/5$, $3c-2 < \log^{1/5} n$ for large enough $n$. Therefore we can apply the fast fractional cascading technique to interconnect the lists according to the topology of the virtual forest so that we can search each list in constant time after the initial search at the root of $F$ without increasing the space requirements.

In summary, handling a query consists of processing the nodes on a path from the root to a leaf node. Processing the root $w$ of $T$ takes $O(\log n/\log\log n + f(w))$ time. The time spent processing any other internal node $u$ is $O(f(u))$. To search the leaf node $w$, we simply check each segment stored there. Since there are at most $O(c)$ such segments and $c = \log^\epsilon n < \log n/\log\log n$ for large enough $n$, the overall query time is $O(\log n/\log\log n)$, and therefore we have the following theorem.

THEOREM 4.5. *There exists a linear-space algorithm to handle the orthogonal segment intersection problem in $O(\log n/\log\log n + f)$ query time, where $f$ is the number of segments reported.*

## 5. Rectangle point enclosure.

To simplify our presentation, we assume that the corners of the rectangles in $S$ all have distinct $x$- and $y$-coordinates. As in the case of the segment intersection problem, the primary data structure consists of a tree $T$ of degree $c = \log^\epsilon n$. Let $v$ be the root of $T$ and $b_1(v), b_2(v), \ldots, b_{c-1}(v)$ be a set of infinite vertical lines, called the *boundaries* of $v$, which partition the set of $2n$ vertical edges of the rectangles in $S$ into $c$ subsets of equal size, thereby creating $c$ stripes $P_1(v), P_2(v), \ldots, P_c(v)$. We define the $c$ subtrees rooted at the children of $v$ recursively, each with respect to the vertical edges that fall into the same stripe. If the number of vertical edges is less than $\log^\epsilon n$ in a stripe, the child node corresponding to this stripe becomes a leaf node. Clearly the height of this tree is $O(\log n/\log\log n)$. A Q-heap $Q(v)$ holding the boundaries of $v$ is built for each node $v$, which will enable a constant time identification of the stripe of $v$ to which the query point belongs. In addition, for each node $v$, except for the root, we define its *x-range* as the stripe $P_i(u)$ of its parent $u$, assuming $v$ is the $i$th child of $u$ from the left.

We associate with each internal node $v$ the rectangles that intersect at least one of its boundaries but none of the boundaries of its ancestors. Each leaf node contains the set of rectangles both of whose vertical edges lie within its $x$-range. Hence at most $O(\log^\epsilon n) = O(\log n/\log\log n)$ rectangles are associated with a leaf node, and no preprocessing will be required for these rectangles.

Now consider the set $S(v)$ of rectangles associated with an internal node $v$. As in [3], we build two hive-graphs $HL_i(v)$ and $HR_i(v)$, as defined in section 2.3, for each boundary $b_i(v)$, $i = 1, \ldots, c-1$. The hive-graph $HL_i(v)$ is built on left vertical edges lying inside stripe $P_i$ and is used to answer the semi-infinite segment intersection queries of the form $(x_1 \leq x, y_1 \leq y \leq y_2)$; $HR_i(v)$ is built on the right vertical edges lying inside stripe $P_{i+1}$ and is used to answer the semi-infinite segment intersection queries of the form $(x_2 \geq x, y_1 \leq y \leq y_2)$. In addition, let $S'(v)$ be a subset of $S(v)$ such that each rectangle in $S'(v)$ crosses at least two boundaries. We transform the coordinates of $S'(v)$ from the space $\mathcal{N} \times \mathcal{N}$ to $W(v)$ in the space $[1, 2, \ldots, c-1] \times \mathcal{N}$ as follows. We transform each rectangle $[x_1, x_2; y_1, y_2]$ in $S'(v)$ into the rectangle $[k_1, k_2; y_1, y_2]$ in $W(v)$, where $b_{k_1}(v)$ and $b_{k_2}(v)$ are the leftmost and rightmost boundaries it crosses.

We now turn our attention to the query algorithm and postpone the description of the data structure for $W(v)$ until the end of this section. Using the Q-heaps stored at the internal nodes, we can in $O(\log n/\log\log n)$ determine the path from the root to the leaf node whose $x$-range contains the query point $p = (x, y)$. (We assume for simplicity that the $x$- and $y$-coordinates of $p$ are different from those of the corners of the rectangles in $S$.) It is clear that only the rectangles associated with the nodes on this path can possibly contain the query point.

Consider a node $v$ on this path. If $v$ is a leaf node, we simply examine each rectangle associated with it, a process that takes $O(\log n/\log\log n + f(v))$ time, where $f(v)$ denotes the number of rectangles reported at node $v$.

If $v$ is an internal node, we first decide which stripe of $v$ the query point $p$ belongs to, a task that can be done in $O(1)$ time; say it belongs to $P_i(v)$. The rectangles stored at node $v$ and which contain $p$ can be classified into three groups: (i) the set $L(v)$ that contains the rectangles whose left vertical edges lie inside $P_i(v)$; (ii) the set $R(v)$ that contains the rectangles whose right vertical edges lie inside $P_i(v)$; and (iii) the set $F(v)$ consisting of those rectangles whose horizontal edges cross $P_i(v)$ entirely. If $i = 1$, $R(v)$ and $F(v)$ do not exist. Similarly, if $i = c$, $L(v)$ and $F(v)$ do not exist.

By Lemma 2.4, the rectangles that belong to the first two groups can be identified in $O(1)$ time per rectangle reported if we apply the fast fractional cascading technique on the lists of the sorted $y$-coordinates of the vertices of the corresponding hive-graph. For example, we know that each rectangle $(x_1, x_2; y_2, y_2)$ associated with the hive-graph $HL_i(v)$ satisfies $x_2 \geq x$. Therefore, to find rectangles in $L(v)$, we only need to check the criteria: $x_1 \leq x$ and $y_1 \leq y \leq y$. Also note that a proper rectangle can be reported at most once in this process.

The remaining task is to determine the rectangles in group $F(v)$, which requires an additional data structure. We start from the set $W(v)$ consisting of rectangles of the form $(k_1, k_2; y_1, y_2)$, where $k_1$ and $k_2$ are integers between 1 and $c-1$. For each pair of different integers $i < j$ between 1 and $c-1$, we construct a Cartesian tree $C_{i,j}(v)$ consisting of rectangles $(i, j; y_1, y_2)$ in $W(v)$ to answer the two-sided range queries in the form $y_1 \leq y \leq y_2$. Note that the total space is still linear, and the use of the fast fractional cascading technique will enable us to access the appropriate nodes in time proportional to the number of rectangles reported.

However, we still need to resolve the problem of identifying which of these Cartesian trees should be accessed when handling a query. We cannot afford to access such a tree unless we are guaranteed to find at least one proper rectangle. To address this problem, we do the following.

We construct a look-up table $M(v)$ with $n'$ rows, each corresponding to a distinct $y$-coordinate of the horizontal edges of the rectangles in $W(v)$ and occupying one word (of $\log n$ bits). The rows are sorted by increasing order of the $y$-coordinates. Let $y_1(v) < y_2(v) < \cdots < y_{n'}(v)$ be the set of distinct $y$-coordinates of the horizontal edges. Let $V_j(v) = (b_{c^3}, b_{c^3-1}, \ldots, b_1)$ be a sequence of $c^3$ bits, where $b_i$ is the $i$th bit from the lower end of the word representing the $j$th row of $M(v)$ (note that $c^3 < \log n$). The word $V_j(v)$ is evenly divided into $c$ sections, each corresponding to a stripe of $v$ (actually we only use $c-2$ of them which correspond to $P_2(v), \ldots, P_{c-1}(v)$). Let $(b_{l+c^2} b_{l+c^2-1} \cdots b_{l+1})$ be one of them that corresponds to $P_i(v)$, $i = 2, \ldots, c-1$. For each pair of integers $k_1 < i \leq k_2$ between 1 and $c-1$, we set the bit $b_{l+k_1 \cdot c+k_2}$ to 1 if there is a rectangle $(k_1, k_2; y_1, y_2)$ in $R$ such that $y_1 < y_j(v) \leq y_2$. All the other bits are set to 0.

To find the proper rectangles in $S'(v)$, we first transform in $O(1)$ time using $Q(v)$ the query point $(x, y)$ to the point $(k, y)$ in the same space as $W(v)$. Let $b_k(v)$ be the leftmost boundary of $v$ whose $x$-coordinate is greater than or equal to $x$. It is clear that a rectangle $(x_1, x_2; y_1, y_2)$ in $S'(v)$ contains $(x, y)$ if and only if its corresponding rectangle $(k_1, k_2; y_1, y_2)$ in $W(v)$ contains $(k, y)$. Let $y_j(v) = \min\{y_l(v) | 1 \leq l \leq n', y_l(v) \geq y\}$. We have the following lemma.

LEMMA 5.1. *Let $(b_{l+c^2} b_{l+c^2-1} \cdots b_{l+1})$ be the section of $V_j(v)$ which corresponds to $P_k(v)$. Then for each pair of integers $1 \leq k_1 < k_2 \leq c-1$ such that $k_1 < k \leq k_2$, $b_{l+k_1 \cdot c+k_2} = 1$ if and only if there exists a rectangle $(k_1, k_2; y_1, y_2) \in W(v)$ which contains $(k, y)$.*

*Proof.* By the definition of $V_j(v)$, $b_{l+k_1 \cdot c+k_2} = 1$ if and only if there exists a rectangle $(k_1, k_2; y_1, y_2)$ such that $y_1 < y_j(v) \leq y_2$. If this rectangle indeed exists, we have $k_1 \leq k \leq k_2$ and $y_2 \geq y_j(v) \geq y$. The definition of $y_j(v)$ ensures that $y \geq y_1$. Therefore $(k_1, k_2; y_1, y_2)$ contains $(k, y)$. Now suppose there is a $(k_1, k_2; y_1, y_2)$ in $W(v)$ which satisfies $k_1 \leq k \leq k_2$ and contains $(k, y)$. The only scenario in which $(k_1, k_2; y_1, y_2)$ does not satisfy $y_1 < y_j(v) \leq y_2$ is $y_j(v) = y = y_1$. This is not possible, given the assumption that $y$ can not be the $y$-coordinate of any horizontal edge.[1] $\square$

Lemma 5.1 shows that the section $B$ of $V_j(v)$ which corresponds to the stripe containing $(k, y)$ indicates correctly the Cartesian trees in $\{C_{i,j} | 2 \leq i < j \leq c-1\}$ which should be visited. Using a look-up table of size $O(n)$, similar to the one described in [24], we can transform $B$ into a list of integers $(m, I_1, I_2, \ldots, I_m)$, where $m$ is the number of 1-bits in $B$ and $I_l$ is the index of a unique Cartesian $C_{i,j}(v)$, for $l = 1, 2, \ldots, m$. Then we simply visit these Cartesian trees one by one.

Searching the sorted lists associated with nonroot nodes can be done using fast fractional cascading. The correctness proof and complexity analysis for this part is similar to that in section 4 and thus is omitted here.

THEOREM 5.2. *There exists a linear-space algorithm to handle the rectangle point enclosure queries in $O(\log n / \log\log n + f)$ time, where $f$ is the number of segments reported.*

## REFERENCES

[1] M. A. BENDER AND M. FARACH-COLTON, *The LCA problem revisited*, in Proceedings of Latin American Theoretical Informatics, Punta del Este, Uruguay, 2000, Springer-Verlag, London, pp. 88–94.

[2] J. L. BENTLEY, *Multidimensional binary search trees used for associative searching*, Comm. ACM, 18 (1975), pp. 509–517.

[1]This assumption might seem to be crucial to the correctness of the lemma. However, without this assumption, the only case that needs a special care is when $(k, y)$ is contained in no rectangle of the form of $(k_1, k_2; y_1, y_2)$ except those satisfying $y_2 = y$. This can be fixed by modifying $L(v)$ to make sure that this special case is not missed.

[3] B. Chazelle, *Filtering search: A new approach to query-answering*, SIAM J. Comput., 15 (1986), pp. 703–724.

[4] B. Chazelle, *Lower bounds for orthogonal range search* I. *The reporting case*, J. ACM, 37 (1990), pp. 200–212.

[5] B. Chazelle and H. Edelsbrunner, *Linear space data structures for two types of range search*, Discrete Comput. Geom., 3 (1987), pp. 113–126.

[6] B. Chazelle and L. J. Guibas, *Fractional cascading:* I. *A data structure technique*, Algorithmica, 1 (1986), pp. 133–162.

[7] B. Chazelle and L. J. Guibas, *Fractional cascading:* II. *Applications*, Algorithmica, 1 (1986), pp. 163–191.

[8] R. Cole, *Searching and storing similar lists*, J. Algorithms, 7 (1986), pp. 202–220.

[9] H. Edelsbrunner, L. J. Guibas, and J. Stolfi, *Optimal point location in a monotone subdivision*, SIAM J. Comput., 15 (1986), pp. 317–340.

[10] M. L. Fredman and D. E. Willard, *Surpassing the information theoretic bound with fusion trees*, J. Comput. System Sci., 47 (1993), pp. 424–436.

[11] M. L. Fredman and D. E. Willard, *Trans-dichotomous algorithms for minimum spanning trees and shortest paths*, J. Comput. System Sci., 48 (1994), pp. 533–551.

[12] H. N. Gabow, J. L. Bentley, and R. E. Tarjan, *Scaling and related techniques for geometry problems*, in Proceedings of the 16th Annual ACM Symposium on Theory of Computing, Washington, DC, 1984, ACM, New York, pp. 135–143.

[13] D. Harel and R. E. Tarjan, *Fast algorithms for finding nearest common ancestors*, SIAM J. Comput., 13 (1984), pp. 338–355.

[14] J. JaJa, C. W. Mortensen, and Q. Shi, *Space-efficient and fast algorithms for multidimensional dominance reporting and range counting*, in Proceedings of the 15th Annual International Symposium on Algorithms and Computation (ISAAC'04), Hong Kong, China, 2004, Lecture Notes in Comput. Sci. 3341, Springer, New York, pp. 558–568.

[15] D. Kirkpatrick, *Optimal search in planar subdivisions*, SIAM J. Comput., 12 (1983), pp. 28–35.

[16] W. Lipski and F. P. Preparata, *Segments, rectangles, contours*, J. Algorithms, 2 (1981), pp. 63–76.

[17] R. J. Lipton and R. E. Tarjan, *Applications of a planar separator theorem*, SIAM J. Comput., 9 (1980), pp. 615–627.

[18] C. Makris and A. K. Tsakalidis, *Algorithms for three-dimensional dominance searching in linear space*, Inform. Process. Lett., 66 (1998), pp. 277–283.

[19] E. M. McCreight, *Priority search trees*, SIAM J. Comput., 14 (1985), pp. 257–276.

[20] K. Mehlhorn, *Data Structures and Algorithms* 3: *Multi-Dimensional Searching and Computational Geometry*, Springer-Verlag, New York, Berlin, 1984.

[21] S. Ramaswamy and S. Subramanian, *Path caching*: *A technique for optimal external searching*, in Proceedings of the ACM Symposium on Principles of Database Systems, San Francisco, CA, 1994, ACM, New York, pp. 25–35.

[22] N. Sarnak and R. E. Tarjan, *Planar point location using persistent search trees*, Comm. ACM, 29 (1986), pp. 669–679.

[23] R. Seidel and C. R. Aragon, *Randomized search trees*, Algorithmica, 16 (1996), pp. 464–497.

[24] Q. Shi and J. JaJa, *Fast algorithms for* 3*-d dominance reporting and counting*, Internat. J. Found. Comput. Sci., 15 (2004), pp. 673–684.

[25] S. Subramanian and S. Ramaswamy, *The P-range tree*: *A new data structure for range searching in secondary memory*, in Proceedings of the ACM-SIAM Symposium on Discrete Algorithms (SODA), San Francisco, 1995, SIAM, Philadelphia, pp. 378–387.

[26] V. K. Vaishnavi and D. Wood, *Rectilinear line segment intersection, layered segment trees and dynamization*, J. Algorithms, 3 (1982), pp. 160–176.

[27] V. K. Vaishnavi, *Computing point enclosures*, IEEE Trans. Comput., C-31 (1982), pp. 22–29.

[28] J. Vuillemin, *A unifying look at data structures*, Comm. ACM, 23 (1980), pp. 229–239.

[29] D. E. Willard, *New data structure for orthogonal range queries*, SIAM J. Comput., 14 (1985), pp. 232–253.

[30] D. E. Willard, *Examining computational geometry, van Emde Boas trees, and hashing from the perspective of the fusion tree*, SIAM J. Comput., 29 (2000), pp. 1030–1049.

SIAM J. COMPUT.
Vol. 34, No. 6, pp. 1493–1515

# COMPLEXITIES FOR GENERALIZED MODELS OF SELF-ASSEMBLY[*]

GAGAN AGGARWAL[†], QI CHENG[‡], MICHAEL H. GOLDWASSER[§], MING-YANG KAO[¶], PABLO MOISSET DE ESPANES[||], AND ROBERT T. SCHWELLER[¶]

**Abstract.** In this paper, we study the complexity of self-assembly under models that are natural generalizations of the tile self-assembly model. In particular, we extend Rothemund and Winfree's study of the tile complexity of tile self-assembly [*Proceedings of the* 32*nd Annual ACM Symposium on Theory of Computing*, Portland, OR, 2000, pp. 459–468]. They provided a lower bound of $\Omega(\frac{\log N}{\log\log N})$ on the tile complexity of assembling an $N \times N$ square for almost all $N$. Adleman et al. [*Proceedings of the* 33*rd Annual ACM Symposium on Theory of Computing*, Heraklion, Greece, 2001, pp. 740–748] gave a construction which achieves this bound. We consider whether the tile complexity for self-assembly can be reduced through several natural generalizations of the model. One of our results is a tile set of size $O(\sqrt{\log N})$ which assembles an $N \times N$ square in a model which allows flexible glue strength between nonequal glues. This result is matched for almost all $N$ by a lower bound dictated by Kolmogorov complexity. For three other generalizations, we show that the $\Omega(\frac{\log N}{\log\log N})$ lower bound applies to $N \times N$ squares. At the same time, we demonstrate that there are some other shapes for which these generalizations allow reduced tile sets. Specifically, for thin rectangles with length $N$ and width $k$, we provide a tighter lower bound of $\Omega(\frac{N^{1/k}}{k})$ for the standard model, yet we also give a construction which achieves $O(\frac{\log N}{\log\log N})$ complexity in a model in which the temperature of the tile system is adjusted during assembly. We also investigate the problem of verifying whether a given tile system uniquely assembles into a given shape; we show that this problem is NP-hard for three of the generalized models.

**Key words.** Kolmogorov complexity, polyominoes, self-assembly, tile complexity, tilings, Wang tiles

**AMS subject classifications.** 05B45, 05B50, 52C20, 52C45, 68Q17, 68Q25, 68Q30

**DOI.** 10.1137/S0097539704445202

## 1. Introduction.

Self-assembly is the ubiquitous process by which objects autonomously assemble into complexes. This phenomenon is common in nature and yet is poorly understood from a mathematical perspective. It is believed that self-assembly technology will ultimately permit the precise fabrication of complex nanostructures. Of particular interest is DNA self-assembly. Double and triple crossover DNA molecules have been designed that can act as four-sided building blocks for

[*]Received by the editors July 16, 2004; accepted for publication (in revised form) March 5, 2005; published electronically August 17, 2005. A preliminary form of this paper appeared in *Proceedings of the* 15*th Annual ACM-SIAM Symposium on Discrete Algorithms*, New Orleans, LA, 2004, SIAM, Philadelphia, 2004, pp. 873–882.

http://www.siam.org/journals/sicomp/34-6/44520.html

[†]Department of Computer Science, Stanford University, Stanford, CA 94305 (gagan@cs.stanford.edu). This author's research was supported in part by a Stanford Graduate Fellowship, NSF grant ITR-0331640, and NSF grant EIA-0137761.

[‡]Department of Computer Science, University of Oklahoma, Norman, OK, 73019 (qcheng@ou.edu). This author's research was supported by an NSF CAREER award.

[§]Department of Mathematics and Computer Science, Saint Louis University, 220 North Grand Blvd., St. Louis, MO 63103 (goldwamh@ slu.edu).

[¶]Department of Computer Science, Northwestern University, Evanston, IL 60201 (kao@ cs.northwestern.edu, schwellerr@cs.northwestern.edu). The research of these authors was supported in part by NSF grant EIA-0112934.

[||]Department of Computer Science, University of Southern California, Los Angeles, CA 90089 (pmoisset@usc.edu). This author's research was supported in part by a grant from NASA/JPL.

DNA self-assembly [4, 5]. Experimental work has been done to show the effectiveness of using these building blocks to assemble DNA crystals and perform DNA computation [6, 8, 12, 13]. With these building blocks (tiles) in mind, researchers have considered the power of the *tile* self-assembly model.

The tile assembly model extends the theory of Wang tilings of the plane [10] by adding a natural mechanism for growth. Informally, the model consists of a set of four-sided Wang tiles whose sides are each associated with a type of glue. The bonding strength between any two glues is determined by a *glue function.* A special tile in the tile set is called the *seed* tile. Assembly takes place by starting with the seed tile and attaching copies of tiles from the tile set one-by-one to the growing seed whenever the total strength of attraction from the glue function meets or exceeds a fixed parameter called the *temperature.*

In this paper we focus on two fundamental complexities relating to this model. The first is the *tile complexity* of self-assembled shapes, defined as the minimum number of distinct Wang tiles required to assemble the shape. Rothemund and Winfree [9] introduced this concept and considered the tile complexity of self-assembled squares. The second is the complexity of verifying whether or not a system of tiles uniquely assembles into a given shape. This problem is considered by Adleman et al. [2] and shown to have a polynomial time solution for the standard tile assembly model.

We extend these studies of tile complexity and shape verification by considering the removal of various restrictions the standard model places on the assembly process. From this we obtain four natural generalizations of the tile assembly model that exhibit significant differences in tile complexity and the complexity of shape verification from that of the standard model. This suggests that some previously derived bounds on tile self-assembly are artifacts of artificial restrictions placed on the standard tile assembly model, rather than bounds on the actual process of self-assembly. Our results are summarized in Table 1.1. Informally, the generalized models are defined as follows.

In the *flexible glue* model, the restriction imposed by Rothemund and Winfree [9], that differing glue types have bonding strength of zero, is removed. In the *multiple temperature* model, the temperature of the system is permitted to change during the assembly process. In the *multiple tile* model, tiles may cooperate by assembling into supertiles before being added to the growing result. In the *unique shape* model, a system *uniquely assembles* a shape if the shape of its resultant supertiles is unique, though its produced supertile may not be unique.

In the standard tile assembly model, Kolmogorov complexity dictates a lower bound of $\Omega(\frac{\log N}{\log\log N})$ on the tile complexity of self-assembling $N \times N$ squares for almost all values of $N$ [9]. Adleman et al. [1] showed how to reach this bound for all $N$. We show (in Theorem 6.1) that this lower bound also applies to each of the generalized models except for the flexible glue model. For this model, Kolmogorov complexity dictates only an $\Omega(\sqrt{\log N})$ lower bound for almost all $N$ (as shown in Theorem 6.2). We show how to achieve this bound for all $N$ by encoding an arbitrary $(\log N)$-bit binary number into the glue function (Theorem 5.1).

Additionally, for some of the models we provide a lower bound of $\Omega(\frac{N^{\frac{1}{k}}}{k})$ (see Theorem 3.1) for assembling *thin* width $k$ length $N$ rectangles (a $k \times N$ rectangle is *thin* if $k < \frac{\log N}{\log\log N - \log\log\log N}$). These lower bounds show that it can require significantly larger tile sets to assemble thinner rectangles than thicker rectangles. With this in mind, we utilize the multiple temperature model to construct a thin rectangle by first constructing a thicker rectangle using a small tile set. We then raise the temperature of the system so that portions of the larger rectangle fall apart,

Table 1.1

*This table gives the general flavor of our results regarding upper and lower asymptotic bounds on tile complexity under the generalized models for $k \times N$ rectangles, as well as the complexity of the* Unq-Shape *problem, but the reader should reference precise details in the stated theorems. A rectangle is* thin *when $k < \frac{\log N}{\log\log N - \log\log\log N}$ and* thick *otherwise. NP-X denotes the class of decision problems not in NP or co-NP unless P = NP.*

| Complexity of self-assembling $k \times N$ rectangles, $k \le N$, and shape verification | | | | | |
|---|---|---|---|---|---|
| | Thin rectangles LB | Thin rectangles UB | Thick rectangles LB | Thick rectangles UB | Unq-Shape |
| Standard | $\frac{N^{\frac{1}{k}}}{k}$ (Thm. 3.1) | $N^{\frac{1}{k}}$ (Thm. 3.2) | $\frac{\log N}{\log\log N}$ (see [9]) | (see [1]) | P (see [2]) |
| Flexible glue | $\frac{N^{\frac{1}{k}}}{k}$ (Thm. 3.1) | $N^{\frac{1}{k}}$ (Thm. 3.2) | $\sqrt{\log N}$ (Thm. 6.2) | (Thm. 5.1) | P (see [2]) |
| Multiple temperature | $\frac{\log N}{\log\log N}$ (Thm. 6.1) | (Thm. 4.1) | $\frac{\log N}{\log\log N}$ (Thm. 6.1) | (see [1]) | NP-X (Thm. 7.6) |
| Multiple tile | $\frac{\log N}{\log\log N}$ (Thm. 6.1) | $N^{\frac{1}{k}}$ (Thm. 3.2) | $\frac{\log N}{\log\log N}$ (Thm. 6.1) | (see [1]) | NP-X (Thm. 7.7) |
| Unique shape | $\frac{N^{\frac{1}{k}}}{k}$ (Thm. 3.1) | $N^{\frac{1}{k}}$ (Thm. 3.2) | $\frac{\log N}{\log\log N}$ (Thm. 6.1) | (see [1]) | Co-NPC (Thm. 7.1) |

leaving the desired rectangle (Theorem 4.1).

Given a tile system, Adleman et al. [2] give an algorithm to verify whether the tile system uniquely assembles into a given supertile. However, in the unique shape model, a tile system could uniquely assemble into a shape even if it does not produce a unique supertile. We investigate the problem of verifying whether a given tile system uniquely assembles into a given shape in the unique shape model, and show that this problem is co-NP-complete (Theorem 7.1). We also show that the problem of determining whether a tile system uniquely assembles into a given supertile is not in NP or co-NP for the multiple temperature model (Theorem 7.6) or the multiple tile model (Theorem 7.7) unless P = NP. These results are in contrast to the polynomial-time verification algorithm for the standard model [2].

*Paper layout.* In section 2 the standard tile model is defined, as well as the generalized models. In section 3 we introduce a construction for assembling $k \times N$ rectangles and provide a lower bound on the tile complexity of such rectangles. In section 4 we use this construction to show how the multiple temperature model can reduce tile complexity when $k \ll N$. In section 5 we use the flexible glue model to reduce the tile complexity of assembling $N \times N$ squares from $\Theta(\frac{\log N}{\log\log N})$ to $\Theta(\sqrt{\log N})$. In section 6 we discuss the lower bounds for assembling shapes of extent $N$ for the generalized models as dictated by Kolmogorov complexity. Finally, in section 7, we show that shape verification is co-NP-complete in the unique shape model and not in NP or co-NP unless P = NP for the multiple temperature and multiple tile models.

## 2. Basics.

**2.1. The standard model.** For alternate descriptions of the tile assembly model, see [1, 2, 9, 11]. Briefly, a tile $t$ in the model is a four-sided Wang tile denoted by the ordered quadruple $(\text{north}(t), \text{east}(t), \text{south}(t), \text{west}(t))$. The entries of the quadruples are glue types taken from an alphabet $\Sigma$ representing the north, east, south, and west edges, respectively, of the Wang tile. A *tile system* is an ordered quadruple $\langle T, s, G, \tau \rangle$, where $T$ is a set of tiles called the *tileset* of the system, $\tau$ is a positive integer called the *temperature* of the system, $s \in T$ is a single tile called the *seed* tile, and $G$ is a function from $\Sigma^2$ to $\{0, 1, \ldots, \tau\}$ called the *glue function* of the system. It is assumed that $G(x, y) = G(y, x)$, and there exists a $\texttt{null}$ in $\Sigma$ such that $\forall x \in \Sigma$, $G(\texttt{null}, \texttt{x}) = \texttt{0}$. In the standard tile assembly model [1, 2, 9], the glue function is such that $G(x, y) = 0$ when $x \neq y$. When dealing with the standard glue model, denote $G(x, x)$ by $G(x)$.

Define a *configuration* to be a mapping from $\mathbb{Z}^2$ to $T \bigcup \{\texttt{empty}\}$, where $\texttt{empty}$ is a special tile that has the $\texttt{null}$ glue on each of its four edges. The *shape* of a configuration is defined as the set of positions $(i, j)$ that do not map to the empty tile. For a configuration $C$ a tile $t \in T$ is said to be *attachable* at the position $(i, j)$ if $C(i, j) = \texttt{empty}$ and $G(\text{north}(t), \text{south}(C(i, j+1))) + G(\text{east}(t), \text{west}(C(i+1, j))) + G(\text{south}(t), \text{north}(C(i, j-1))) + G(\text{west}(t), \text{east}(C(i-1, j))) \geq \tau$. For configurations $C$ and $C'$ such that $C(x, y) = \texttt{empty}$, $C'(i, j) = C(i, j)\ \forall (i, j) \neq (x, y)$ and $C'(x, y) = t$ for some $t \in T$ define the act of *attaching* tile $t$ to $C$ at position $(x, y)$ as the transformation from configuration $C$ to $C'$.

Define the *adjacency graph* of a configuration $C$ as follows. Let the set of vertices be the set of coordinates $(i, j)$ such that $C(i, j)$ is not empty. Let there be an edge between vertices $(x_1, y_1)$ and $(x_2, y_2)$ iff $|x_1 - x_2| + |y_1 - y_2| = 1$. We refer to a configuration whose adjacency graph is finite and connected as a *supertile*. For a supertile $S$, denote the number of nonempty positions (tiles) in the supertile by $\text{size}(S)$. We also note that each tile $t \in T$ can be thought of as denoting the unique supertile that maps position $(0, 0)$ to $t$ and all other positions to $\texttt{empty}$. Throughout this paper we will informally refer to tiles as being supertiles.

A *cut* of a supertile is a cut of the adjacency graph of the supertile. In addition, for each edge $e_i$ in a cut define the *edge strength* $s_i$ of $e_i$ to be the glue strength (from the glue function) of the glues on the abutting edges of the adjacent tiles corresponding to $e_i$. Now define the *cut strength* of a cut $c$ to be $\sum s_i$ for each edge $e_i$ in the cut.

In the standard model, assembly takes place by *growing* a supertile starting with tile $s$ at position $(0, 0)$. Any $t \in T$ that is attachable at some position $(i, j)$ may attach and thus increase the size of the supertile. For a given tile system, any supertile that can be obtained by starting with the seed and attaching arbitrary attachable tiles is said to be *produced*. If this process comes to a point at which no tiles in $T$ can be added, the resultant supertile is said to be *terminally* produced. For a given shape $\Upsilon$, a tile system $\Gamma$ *uniquely produces* shape $\Upsilon$ if there exists a terminal supertile $S$ of shape $\Upsilon$ such that every supertile derived from the seed can be grown into $S$. The *tile complexity* of a shape $\Upsilon$ is the minimum tile set size required to uniquely assemble $\Upsilon$ under a given assembly model. While the definition of unique shape production provided here is the same as in [1, 2, 9], we note that it is somewhat unnatural in that it requires unique production of supertiles. In section 2.2 we introduce a model that utilizes a more natural definition of unique shape production.

**2.2. Four generalized models.**

*The multiple temperature model.* In the *multiple temperature* model, the integer temperature $\tau$ in the tile system description is replaced with a sequence of integers

$\{\tau_i\}_{i=1}^{k}$ called the *temperature sequence* of the system. A system with $k$ temperatures in its temperature sequence is said to be a *$k$-temperature* system.

In a $k$-temperature system, assembly takes place in $k$ phases. In the first phase, assembly takes place as in the standard model under temperature $\tau_1$. Phase 1 continues until no tiles can be added. In phase two, tiles can be added or removed under $\tau_2$. Specifically, at any point during phase 2 if there exists a cut of the resultant supertile with cut strength less than $\tau_2$, the portion of the supertile occurring on the side of the cut not containing the seed tile may be removed. Also, at any point in the second phase any tile in $T$ may be added to the supertile if the tile is attachable at a given position under temperature $\tau_2$. The second phase of this assembly continues until no tiles can be added or removed. We then go to phase 3 in which tiles may be added and removed under temperature $\tau_3$. The process is continued up through $\tau_k$. For any given choice of additions and removals such that each of the $k$ phases finishes, the tile system *terminally* produces the corresponding shape assembled after phase $k$. If the $k$ phases always finish regardless of the choice of additions and removals, and the terminally produced supertile $R$ is unique, the tile system *uniquely assembles* the shape of $R$ under the $k$-temperature model.

*The flexible glue model.* In the *flexible glue* model, the restriction that $G(x, y) = 0$ for $x \neq y$ is removed.

*The unique shape model.* In the *unique shape* model, we redefine what it means for a system to uniquely produce a shape $S$. In this model, a tile system uniquely produces a shape $S$ iff all producible supertiles can be grown into a terminal supertile with shape $S$. Thus, the system may produce many different terminal supertiles as long as they all have the desired shape.

*The multiple tile model.* In the *multiple tile* model with parameter $q$, tiles can combine into supertiles of size at most $q$ before being added to the growing seed supertile. Attaching a supertile is much like attaching a singleton tile. Intuitively, two supertiles attach to one another by abutting together and sticking if the total edge strength of the abutting edges meets or exceeds the temperature $\tau$. However, the formal definition of a supertile is a connected configuration of tiles on $\mathbb{Z}^2$. This definition makes a distinction between two supertiles that are equal up to translation. However, we actually wish to think of two such supertiles as the same supertile. We therefore use the following formal definition of supertile attachment, which generalizes the notion of singleton tile attachment.

For any two supertiles $A$ and $B$, define $A \oplus B$ to be the configuration obtained by setting all empty positions $A(i, j)$ equal to $B(i, j)$. Further, we say $A \oplus B$ is *valid* if it is a supertile (i.e., it is connected), and if $\forall\,(i, j)$ either $A(i, j) = \texttt{empty}$ or $B(i, j) = \texttt{empty}$. For a valid supertile $A \oplus B$, denote the cut defined by separating all the tile positions initially in $A$ from all the tile positions initially in $B$ as the *border* cut. For a supertile $A$ and integers $x$ and $y$, let $A_{x,y}$ denote the configuration obtained by translating $A$ by $x$ units along the $x$-axis and $y$ units along the $y$-axis. Finally, for a given temperature $\tau$, supertile $B$ can be attached to supertile $A$ to form supertile $A \oplus B_{x,y}$ if $A \oplus B_{x,y}$ is valid and the cut strength of the border of $A \oplus B_{x,y}$ is at least $\tau$.

Having defined how to attach supertiles, we now define multiple tile systems. A tile system in the multiple tile model is a quintuple $\langle T, s, G, \tau, q \rangle$. Intuitively, in a multiple tile system any supertile of size at most $q$ that can be obtained by combining two smaller addable supertiles is also an addable supertile. The assembly process then permits any addable supertile to attach to the growing seed supertile if the tile can be placed so that the abutting edges of the two supertiles have a total strength that meets or exceeds the temperature $\tau$. For such a system define a set of *addable* supertiles $W_T$ as follows. First, $T \subseteq W_T$. Second, if there exist $A, B \in W_T$ such that $\text{size}(A) + \text{size}(B) \leq q$ and $B$ can be attached to $A$ under temperature $\tau$ to obtain $A \oplus B_{x,y}$, then $A \oplus B_{x,y}$ is also in $W_T$. In the multiple tile assembly model, supertiles from the set $W_T$ as well as from $T$ may be added. Specifically, any supertile that is in the set of addable supertiles $W_T$ that is attachable to the seed supertile under temperature $\tau$ can attach and increase the size of the seed supertile. For $q = 1$, we have $W_T = T$, which gives us the standard model of assembly.

**3. The assembly of $k \times N$ rectangles.** In this section, we present both an upper and a lower bound for assembling $k \times N$ rectangles for arbitrary $k$ and $N$. The upper bound comes from a simple construction which constitutes a $k$-digit base-$N^{\frac{1}{k}}$ counter and has tile complexity $\Theta(N^{\frac{1}{k}} + k)$. In later sections, we use modifications of this counter to show how both the two-temperature and flexible glue model can reduce tile complexity of certain shapes. Additionally, for the case of $k = \log N$, this construction constitutes a $\log N$-bit base-2 counter which assembles a $\log N \times N$ block. The tile set created by this special case is used in [3] to assemble a $\log N \times N$ block in optimal *time complexity* (as defined in [1]) of $\Theta(N)$. This permits the assembly of $N \times N$ squares in optimal tile complexity and optimal time complexity as in [1], but does so in a much simpler fashion and uses only temperature $\tau = 2$.

THEOREM 3.1. *The tile complexity of self-assembling a $k \times N$ rectangle is $\Omega(\frac{N^{\frac{1}{k}}}{k})$ for the standard model, the unique shape model, and the flexible glue model.*

*Proof.* Suppose there exists a tile system with fewer than $(\frac{N}{2(k!)})^{\frac{1}{k}}$ distinct tile types, which uniquely assembles a $k \times N$ rectangle. Then there must be fewer than $\frac{N}{2(k!)}$ distinct $k$-tile columns consisting of these tiles. So in a $k \times N$ configuration, there must exist some $k$-tile sequence which is repeated for more than $2(k!)$ of the columns. And under the standard, unique shape, and flexible glue models, for each of these identical columns we can assign an ordering on the $k$ tiles that corresponds to a possible relative order in which the $k$ tiles of the given column could be attached. (In the multiple tile model, groups of tiles may be attached simultaneously, and thus this ordering is not sufficient.) Since there are at most $k!$ orderings possible, we get that at least three of the identical columns must also have an identical order of attachment. From this we derive a contradiction as follows. Wherever the seed tile of the construction occurs, it lies to either the west or east of two of the identically placed columns. And if the seed tile occurs to the west (respectively, east) of a column, then all tiles east (respectively, west) of the column are determined by (1) the tiles and their positions in the column, and (2) the relative attachment order of the tiles in the column. That is, if a rectangle has fewer than $(\frac{N}{2(k!)})^{\frac{1}{k}} = \Omega(\frac{N^{\frac{1}{k}}}{k})$ distinct tile types, there necessarily are at least two identical columns east (respectively, west) of the seed tile whose order of attachment is the same. When attaching only a single tile at a time, this guarantees the possibility of producing an infinite repeating loop of tiles. Thus, no finite shape can be uniquely produced. $\square$

*Remark.* We also point out that the argument for Theorem 3.1 applies to any length $N$ shape whose height at each column is bounded by a value $k$.

THEOREM 3.2. *The tile complexity of self-assembling a $k \times N$ rectangle is $O(N^{\frac{1}{k}} + k)$ for the standard model, the unique shape model, and the multiple tile model.*

We show this by first providing a construction for the standard model and then arguing that the construction works for the unique shape model and the multiple tile model as well.

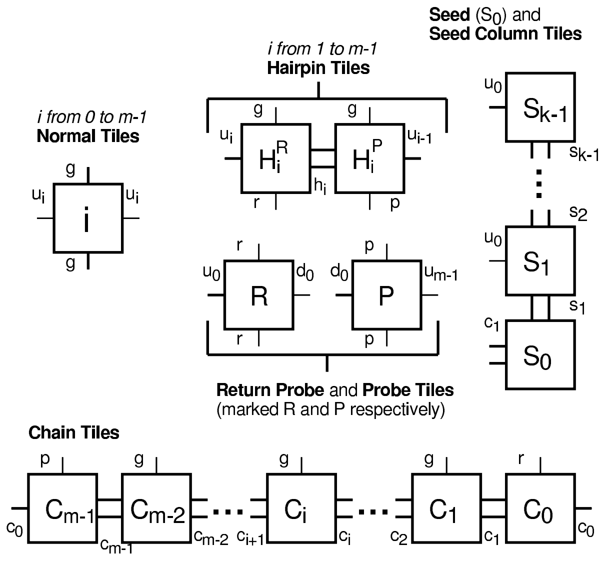


FIG. 1. *This tile set assembles a $k \times m^k$ rectangle in $\Theta(k+m)$ tile complexity under the standard assembly model.*

*Proof for the standard model.* For a given $N$, let $m = \lceil N^{\frac{1}{k}} \rceil$. To show this bound, we give a general tile set for assembling a $k \times m^k$ rectangle in $O(N^{\frac{1}{k}} + k)$ tiles under the standard model. We then show how to adjust the tile set so it produces a $k \times N$ rectangle. The tile system we use constitutes a $k$-digit base-$m$ counter. The system has temperature $\tau = 2$, and the tile set and glue strength function are given in Figure 1.

The assembly takes place as follows. The north edge of the seed tile produces a length $k$ *seed column* from the seed column tiles. The west edge of the seed produces a length $m$ *chain* from the chain tiles. The 0-normal tile can then fill in all the rows and columns up until column $m-1$. In this column the $H_1^P$ hairpin tile must be placed. This causes a *hairpin* growth, which causes another length $m$ chain of chain tiles to be placed in the first row. The next section of $m$ columns are then filled with the 0-normal tile in all rows but the second, which will contain the 1-normal tile or a hairpin tile. In general, when the $C_{m-1}$ chain tile is placed, probe tiles are added on top of each other until a row is found that does not consist of the $(m-1)$-normal tile. In such a row the appropriate pair of hairpin tiles are placed, and a downward growing column of return probe tiles are placed until the bottom row is encountered, at which point the $C_0$ chain tile is placed to start the length $m$ chain growth over again in the first row. If no such row is encountered, the assembly finishes. The idea here is that the bottom *chain* row represents the least significant digit of the counter, thus incrementing every column. After each block of $m$ we need to find the most significant digit that requires incrementation, as well as rollover all the trailing digits displaying $m-1$ to 0. The hairpin tiles perform the incrementation, while the probe and return probe tiles perform the rollover.

It is easy to see that for a given $k$ and $m$, this construction assembles into a $k \times m^k$ block. In addition, we can adjust the glue types of the west edges of the seed column and seed tiles to represent any $k$-digit base-$m$ number. By doing this the counter can be made to start at any number between 0 and $m^k - 1$. Thus, we can designate the length of the constructed shape to be any number between 1 and $m^k$. Therefore, to assemble a $k \times N$ block, we use the above tile set with $m = \lceil N^{\frac{1}{k}} \rceil$ and the west edges of the seed and seed columns tiles set to represent the number $m^k - N$. Thus, we can construct a $k \times N$ block in $4m + k = O(N^{\frac{1}{k}} + k)$ tiles. □

*Proof for the unique shape model.* This follows from the construction for the standard model. □

*Proof for the multiple tile model.* Now we argue that the tile set given for the standard model uniquely assembles a $k \times N$ rectangle for the multiple tile model as well. Let $A$ denote the uniquely produced $k \times N$ supertile from the tile set of Figure 1 under the standard model. From the proof for the standard model we have that $A$ is produced under the multiple tile model as well. We thus need to show that the multiple tile model will not produce any shape other than $A$. Denote any produced supertile under the standard model to be a *substructure* of $A$. We thus want to show that any supertile produced by a multiple tile model is a substructure of $A$.

To show this, consider a multiple tile system with tile set given in Figure 1. Consider the set $W$ of addable supertiles and the set $X \subset W$ of supertiles that do not contain the seed tile. For a given $x \in X$, denote $x$ as a *column* if it consists of only a column of seed column tiles, a *chain* if it consists of only a chain of chain tiles, a *hairpin* if it consists of exactly two hairpin tiles, and a *singleton* if $x \in T$. For our argument we show the following three claims.

*Claim* 1. Every supertile in the set $X$ is either a column, chain, hairpin, or singleton.

*Claim* 2. Attaching a column, chain, hairpin, or singleton to a substructure yields a new substructure.

*Claim* 3. Two substructures cannot be attached to one another.

From this we have that by starting off with a single seed tile, which is a substructure, only supertiles that preserve the substructure property may be added. Thus, the multiple tile model can only produce substructures and thus cannot produce anything that the standard model cannot produce. □

*Proof of Claim* 1. Clearly all supertiles in $X$ of size 1 can be classified as either a chain, column, hairpin, or singleton. (They are all singletons, some are chains, some are columns, none are hairpins.) Inductively, suppose all supertiles in $X$ of size less than $k$ can be classified into the above categories. Consider an arbitrary size $k$ supertile in $X$. By definition of $W$ the supertile must be composed of two smaller supertiles. However, we cannot combine a chain with a column, nor a chain with a hairpin, nor a column with a hairpin. And clearly, if a singleton can be added to some other chain, column, hairpin, or singleton, it must yield a column, chain, or hairpin. Similarly, when two chains are combined, they yield another chain, and when two columns are combined, they yield another column. Thus, the size $k$ supertile is either a chain, column, or hairpin. □

*Proof of Claim* 2. If a column is attachable to a substructure, it is attachable solely due to the glue strength of the south glue on the southernmost tile of the column. Instead of placing the column as a single chunk, each tile can be placed one by one under the standard model, starting with the southernmost tile. Thus, the end result is a substructure. The same type of argument applies to chains and hairpins. □

*Proof of Claim* 3. Among the tiles in an arbitrary substructure, no tile is farther north than the northern most tile in the eastmost column. Similarly, no tile is farther west than the westmost tile in the southernmost row, except for possibly a single hairpin tile. Now, since the eastern edges of all tiles in the first column and the southern edges of all tiles in the first row have bonding strength of 0, the only possible way for two substructures to bond together is for there to be an attraction between the north edge of some tile in the first column of one substructure with the south edge of a hairpin tile of the other substructure. But there is no attraction strength between the northern edges of seed column tiles and the southern edges of hairpin tiles, so this is not possible. ∎

## 4. Reducing tile complexity with the two-temperature model.

Now we show how a multiple temperature model can reduce tile complexity for assembling thin rectangles. For a given $k$ and $N$ with $k \ll N$, the idea is to use a modification of the construction from section 3 to build a $j \times N$ rectangle for optimal (bigger than $k$) $j$ such that the top $j-k$ rows are less stable than the bottom $k$. The temperature of the system is then raised to cause all but the bottom $k$ rows to fall apart. We then compare the complexity of this construction with the lower bound for the standard model given in section 3 to show that the 2-temperature model can beat a lower bound for the standard model.

*Minimizing the complexity.* For a given $j$ and $N$, assembling a $j \times N$ rectangle using the construction of Theorem 3.2 yields an upper bound that is a function of $j$. If we are only interested in constructing a rectangle of length $N$ but do not care about the width, we can choose $j$ as a function of $N$ such that the tile complexity is minimized. To do this we choose a value for $j$ that balances the size of $N^{\frac{1}{j}}$ and $j$. For $j = \frac{\log N}{\log\log N - \log\log\log N} = \Theta(\frac{\log N}{\log\log N})$, the term $N^{\frac{1}{j}}$ is $\frac{\log N}{\log\log N}$. Thus, the number of tiles used in the construction for building a $j \times N$ block for such a $j$ is $j + N^{\frac{1}{j}} = \Theta(\frac{\log N}{\log\log N})$, which meets the lower bound dictated by Kolmogorov complexity for almost all $N$ [9]. We use this result in the following theorem.

THEOREM 4.1. *Under the two-temperature model, the tile complexity of self-assembling a $k \times N$ rectangle for an arbitrary integer $k \geq 2$ is $O(\frac{\log N}{\log\log N})$.*

*Proof.* In the case that $k$ exceeds $\lceil \frac{\log N}{\log\log N - \log\log\log N} \rceil$, we can simply use a single-temperature model to build the two perpendicular axes of the rectangle in optimal $O(\frac{\log N}{\log\log N})$ complexity. The addition of a constant number of tiles can then fill in the rest of the rectangle. Otherwise, to reach the bound we use a tile system that assembles a $j \times N$ block for optimal value $j = \lceil \frac{\log N}{\log\log N - \log\log\log N} \rceil$ under temperature $\tau_1$ and breaks down to a smaller $k \times N$ block in the second phase of the two-temperature assembly process. As in the construction from Theorem 3.2, let $m = \lceil N^{\frac{1}{j}} \rceil$. Consider the two-temperature tile system with $\tau_1 = 4$ and $\tau_2 = 6$ and the tile set and glue strength function given in Figure 2.

Under temperature $\tau_1$, this tile system assembles a $j \times N$ block in exactly the same fashion as the single-temperature system from Theorem 3.2. See Figure 3 for an example of the construction. However, for each tile in the top $j-k$ rows other than the seed column tiles, the north and south edges have glue strength 1. Therefore, since no glue in the system exceeds strength 4, we are assured that any cut consisting of one north-south edge and one east-west edge has cut strength less than $\tau_2 = 6$. Therefore, under the two-temperature model each tile in the top $j-k$ rows can be removed one at a time, starting at the northwest corner of the $j \times N$ block. We are then left with the bottom $k$ rows. Since our design ensures that each edge in the bottom $k$ rows has strength 3 or greater, no cut of the bottom $k \times N$ supertile can have cut strength less than 6. Thus, no cut can break up the remaining $k \times N$ block. And since any alternate choice of cuts would only expedite the process to this end, this construction uniquely assembles a $k \times N$ block in $10m + j = O(N^{\frac{1}{j}} + j) = O(\frac{\log N}{\log\log N})$

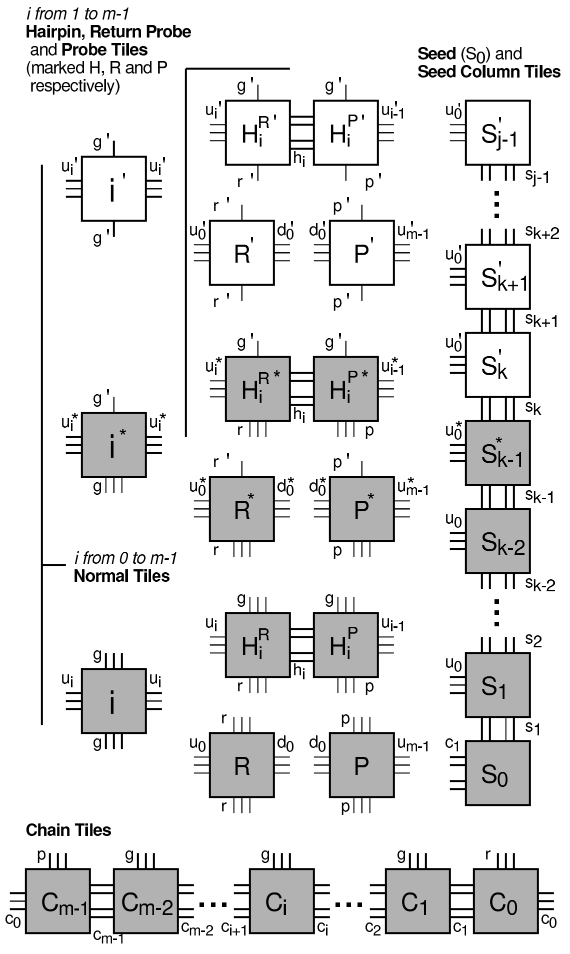

i from 1 to m-1
Hairpin, Return Probe and Probe Tiles (marked H, R and P respectively)
Seed (S0) and Seed Column Tiles
i from 0 to m-1
Normal Tiles
Chain Tiles

FIG. 2. *A tile set to assemble a $k \times N$ rectangle in $\Theta(j+m)$ tile complexity for $m = \lceil N^{\frac{1}{j}} \rceil$ under the two-temperature model with $\tau_1 = 4$ and $\tau_2 = 6$. At temperature $\tau_1$ the shaded tiles fill in the bottom $k$ rows of a $j \times N$ rectangle, while the clear tiles fill the top $j-k$ rows. At temperature $\tau_2$ the clear tiles fall off.*

tile complexity under the two-temperature assembly model. □

For small values of $k$, this beats the lower bound for the standard model. Consider $k = \frac{\log N}{2 \log \log N}$. For such $k$ the value of $N^{\frac{1}{k}}$ is

$$N^{\frac{2 \log \log N}{\log N}} = (\log N)^2.$$

Thus, from Theorem 3.1, the lower bound for the standard model is

$$\Omega\left(\frac{N^{\frac{1}{k}}}{k}\right) = \Omega((\log N)(\log \log N)).$$

Since this bound only gets higher for smaller values of $k$, the two-temperature model yields an asymptotically smaller tile complexity for all $k$ between 2 and $\frac{\log N}{2 \log \log N}$.

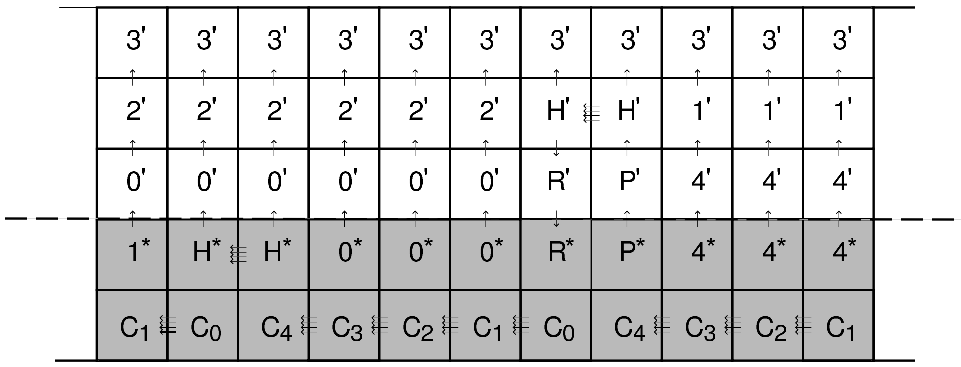


FIG. 3. *Here is a typical section of a self-assembled* $5 \times N$ *block from the tile set in Figure* 2 *that will break down to a* $2 \times N$ *block under temperature* 6*. Edges with strength* 4 *are marked by four adjacent arrows, while edges with strength* 1 *are marked with one. All remaining edges have strength* 3.

**5. Assembling $N \times N$ squares in $O(\sqrt{\log N})$ tiles.** Kolmogorov complexity dictates an $\Omega(\frac{\log N}{\log \log N})$ lower bound for almost all $N$ on the tile complexity of self-assembling $N \times N$ squares for the standard model in which the glue function is limited so that $G(x, y) = 0$ for $x \neq y$. However, if this restriction is lifted, the bound no longer applies. In this section, we show that the tile complexity of self-assembling $N \times N$ squares is $\Theta(\sqrt{\log N})$ under the flexible glue model for almost all $N$.

THEOREM 5.1. *The tile complexity of self-assembling $N \times N$ squares is $O(\sqrt{\log N})$ under the flexible glue model.*

*Proof.* The trick as introduced in [9] is to be able to initialize a fixed length binary counter to any arbitrary $(\log N)$-bit binary number. This can be done trivially in $O(\log N)$ tile complexity. By simulating base conversion, it can be done in $O(\frac{\log N}{\log \log N})$ [1]. Here we provide a tile system that assembles a $2 \times \log N$ block in $O(\sqrt{\log N})$ tiles such that the top row of the block encodes a given binary number $b$. This is accomplished by taking advantage of the flexible glue function and encoding $b$ into the glue function. Let $n = \lfloor \log N \rfloor + 1$ and $m = \lceil \sqrt{n}\, \rceil$. Let $b$ be a given $n$-bit binary number. Let $b_{ij}$ be the bit in position $im + j$ in $b$. We use the tile set and glue function from Figure 4 to construct a $2 \times n$ block such that the top row of the block represents the number $b$. For convenience we denote some glues by the symbols 0 and 1. See Figure 5 for an example of the construction.

To simplify the illustration, the tile set in Figure 4 assumes that $m^2 = n$. If this is not the case, we can do as before in Theorem 3.2 and initialize the 2-digit counter to an arbitrary value $c$ so that a block of length exactly $n$ is assembled. At any rate, the construction uses $5\lceil \sqrt{n}\, \rceil + 1$ tiles and constructs a $2 \times n$ block in the same fashion as the general $k \times n$ assembly. Additionally, the glue function assures us that for the $(i, j)$ digit of $b$, the corresponding position in the top row of the construction can only be tiled with either a hairpin, seed column, or normal tile with north edge glue equal to $b_{ij}$.

To complete the $N \times N$ square we need to create a fixed length binary counter that is initialized to the given binary number $b$ and grows north, incrementing row-by-row, until $2^n$ is reached. The addition of a constant number of stairstep tiles can then finish off the square, as shown in [9]. The fixed length binary counter can also be implemented in a constant number of tiles, as is done in [1, 9]. Alternatively, we

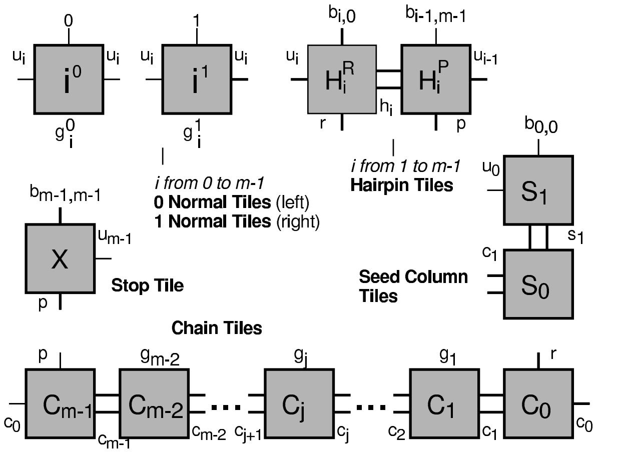


FIG. 4. *This tile set creates a $2 \times n$ block whose top row represents a given $n$-bit binary number $b$. Here $b_{ij}$ is the value of the bit in position $im + j$ in $b$. The glue function for glues $g_i^1$ and $g_j$ for $i$ from $0$ to $m - 1$ and $j$ from $1$ to $m - 2$ is $G(g_i^1, g_j) = b_{ij}$. All other pairs of nonequal glues have strength $0$.*

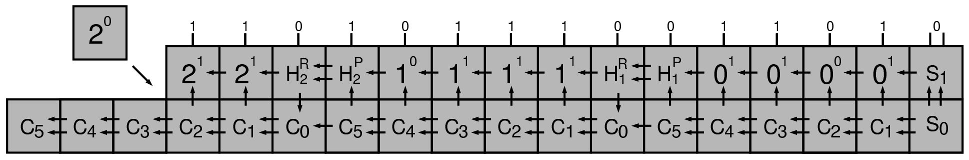


FIG. 5. *Assembling an arbitrary $n$-bit binary number in $O(\sqrt{n})$ tiles. Here we show the construction for $n = 36$ and binary number $b = \ldots 0110101110011010$.*

can use the construction from section 3 for building a $k \times N$ block for $k = \log N$. This yields a binary counter consisting of eight tiles plus $\log N$ seed column tiles. However, since we already have a "seed column" via the $2 \times n$ *seed block*, we can omit the seed column tiles. Thus, the total number of tiles used for the assembly of an $N \times N$ square is only a constant greater than the $\Theta(\sqrt{\log N})$ tiles used to build the $2 \times n$ seed block. $\square$

**6. Kolmogorov lower bounds.** Rothemund and Winfree [9] showed that Kolmogorov complexity dictates an $\Omega(\frac{\log N}{\log \log N})$ lower bound on the tile complexity for self-assembling $N \times N$ squares for almost all $N$. We show that their proof generalizes to the multiple temperature model, the unique shape model, and the multiple tile model. For the flexible glue model, we modify their argument to get an $\Omega(\sqrt{\log N})$ lower bound from Kolmogorov complexity, making the construction in Theorem 5.1 tight for almost all $N$.

THEOREM 6.1. *For almost all $N$ the tile complexity of self-assembling an $N \times N$ square is $\Omega(\frac{\log N}{\log \log N})$ for the multiple tile model, the multiple temperature model, and the unique shape model.*

*Proof.* The Kolmogorov complexity of an integer $N$ with respect to a universal Turing machine $U$ is $K_U(N) = \min|p|$ s.t $U(p) = b_N$, where $b_N$ is the binary representation of $N$. A straightforward application of the pigeonhole principle yields that $K_U(N) \geq \lceil \log N \rceil - \Delta$ for at least $1 - (\frac{1}{2})^{\Delta}$ of all $N$. (See [7] for results on Kolmogorov complexity.) Thus, for any $\epsilon > 0$, $K_U(N) \geq (1 - \epsilon) \log N = \Omega(\log N)$ for almost all $N$.

This tells us that if we have a Turing machine that takes as input a tile system and outputs the maximum length of the shape produced by the given tile system, then the total size in bits of the machine plus the size in bits of a tile system that uniquely assembles an $N \times N$ square is $\Omega(\log N)$.

For a given assembly model, if there exists a finite size Turing machine that takes as input a tile system and outputs the maximum extent (i.e., width or length) of the shape uniquely produced by the system, then the $\Omega(\log N)$ bound applies to the number of bits required to represent the tile system. For each of the nonflexible glue models, let $\tau$ be the maximal temperature of the system and $n$ the size of the tile set. Such tile sets can be represented by $4n \log 4n + 4n \log \tau$ bits as follows. For each of the $4n$ edges in the system assign an index to represent one of the at most $4n$ glues of the system. Also record a mapping from each glue to an index from 0 to $\tau$ representing the corresponding strength of the given glue. Thus, for $\tau$ bounded by a constant, the number of bits required to represent a tile system is $O(n \log n)$, which implies that $n = \Omega(\frac{\log N}{\log \log N})$ for almost all $N$ for any system that self-assembles a shape with maximum extent $N$ [9].

To finish this argument, we explicitly give algorithms for each model, which output the maximum extent of the shape uniquely produced by a given system. For the standard model, the multiple tile model, and the unique shape model, the following algorithm can output the maximum extent of the shape produced by a given tile system.

```
PRODUCED-EXTENT($\mathbf{\Gamma}$).
  1. Set $D_1 = T$. In general, $D_i$ denotes the set of all supertiles of size $i$ which
     consist of tiles in $T$.
  2. Mark each $t \in D_1$ as FEASIBLE.
  3. Set $i = 2$.
  4. Create $D_i$, the set of all supertiles of size $i$. Mark each $t \in D_i$ as FEASIBLE
     if $t$ can be constructed from two smaller FEASIBLE supertiles from
     the set $\bigcup_{r=1}^{i-1} D_r$. If $t$ is FEASIBLE and contains the seed tile, mark $t$ as
     PRODUCED.
  5. Increment $i$ and repeat step 4 unless $\bigcup_{r=\frac{i}{2}}^{i} D_r$ contains no FEASIBLE super-
     tile.
  6. Mark each PRODUCED supertile in $\bigcup_{r=1}^{i} D_r$ as TERMINAL if no FEASI-
     BLE supertile can be attached.
  7. Output the maximum extent (width or length) of any smallest TERMINAL
     supertile in $\bigcup_{r=1}^{i} D_r$.
```

Under the multiple tile model, if there are no addable supertiles of size between $\frac{i}{2}$ and $i$, then we know that there are no addable supertiles of size greater than $i$, since such a tile would need to be assembled from at least one smaller supertile of size at least $\frac{i}{2}$. Thus, in the case where each addable supertile is finite, this algorithm will finish and output the correct result. We also note that the algorithm outputs the extent of the smallest terminally produced supertile for the system. This tells us that the Kolmogorov bound applies to models with even more general definitions of uniqueness than what we consider. For the multiple temperature model we have the following.

```
$k$-TEMPERATURE-PRODUCED-EXTENT($\Gamma$).
  1. Initialize the *current supertile* to be the seed tile.
  2. For $i = 1$ to $k$ do the following. If there exists a cut $c$ in the current supertile
     with strength($c$) $< \tau_i$, remove the tiles on the side of the cut not containing
```

the seed. If there exists some $x \in T$ such that $x$ is addable under $\tau_i$, add $x$. Continue adding and removing until nothing can be added or removed at temperature $\tau_i$.

3. Output the maximum extent of the resultant shape.

Since the definition of the $k$-temperature model requires that a system that uniquely produces a shape must, regardless of the sequence of additions and removals, finish each phase, we are assured that each iteration of the for-loop will finish, thus giving the end result. ∎

Theorem 6.2. *For almost all $N$ the tile complexity of self-assembling an $N \times N$ square is $\Omega(\sqrt{\log N})$ under the flexible glue model.*

*Proof.* The Produced-extent algorithm given in Theorem 6.1 for the standard assembly model works just as well for the flexible glue model. Thus, as discussed in Theorem 6.1, the number of binary digits required to represent a flexible glue tile system that self-assembles an $N \times N$ square is bounded below by $\Omega(\log N)$ for almost all $N$. We can represent a flexible glue tile system of size $n$ tiles and temperature $\tau$ bounded by a constant with $f(n) = 4n\log(4n) + (4n)^2 \log\tau = O(n^2)$ bits as follows. For each tile there are four sides for which we must denote a glue. We need at most $4n$ distinct glues, and for each glue we need to store the bonding strength with every other glue in the model. This can be stored in a $4n \times 4n$ matrix. Now for a given $N$, let $\beta(N)$ be the cardinality of the minimum tile set for assembling an $N \times N$ square under the flexible glue model. Then, there exist constants $C_1$ and $C_2$ such that for almost all $N$,

$$C_1 \log N \leq f(\beta(N)) \leq C_2 \beta(N)^2.$$

Thus $\beta(N) = \Omega(\sqrt{\log N})$ for almost all $N$. ∎

## 7. Shape verification.

Given a shape, Adleman et al. [2] studied the problem of finding the minimum set of tiles which uniquely produces a supertile $A$ such that $A$ has the given shape. To show that the decision version of the problem is in NP, they gave an algorithm to verify whether a given set of tiles uniquely produces a supertile which has the given shape. In contrast, in this section we show that for the unique shape, multiple tile, and multiple temperature models this problem is NP-hard. The problem of shape verification is defined as follows.

Unq-Shape$(\Gamma, M)$ is the problem of deciding whether the tile system $\Gamma$ uniquely assembles into the shape $M$.

### 7.1. Unique shape model.

The algorithm of [2] to verify whether a given set of tiles uniquely produces a given shape insists that the tile system should assemble into a unique terminal supertile. Under the *unique shape* model, a tile system $\mathbb{T}$ uniquely produces a shape if all terminal supertiles of $\mathbb{T}$ (henceforth called $Term(\mathbb{T})$) have the given shape, and no larger supertiles are produced. We note that this definition automatically implies that if a tile system uniquely produces a shape, then $Term(\mathbb{T})$ is nonempty. In this section, we show that the unique shape problem is co-NP-complete under the unique shape model, in contrast to the standard model.

Theorem 7.1. Unq-Shape *is co-NP-complete under the unique shape model. It remains NP-hard even when the temperature $\tau = 2$.*

We begin by showing that Unq-Shape is in co-NP, and then show that it is NP-hard (and thus co-NP-hard).

Lemma 7.2. Unq-Shape *is in co-NP under the unique shape model.*

*Proof.* For this, we need to show that if an instance of Unq-Shape is false, i.e., if the tile system in the instance does not assemble uniquely into the given shape, then there is a short proof of the fact. By definition, a given tile system $\mathbb{T} = \langle T, \mathcal{S}, G, \tau \rangle$ does not uniquely assemble into the given shape $M$ iff one of the following occurs:

1. A terminal supertile of a shape different from $M$ can be assembled. In this case, $Term(\mathbb{T})$ contains a supertile $A$ with a shape different from $M$. Then $A$, along with the order in which the tiles join to assemble $A$, would suffice as a proof. In order to check this proof, we first verify that $A$ can indeed be assembled by adding the tiles in the order specified. Then we check that $A$ is a terminal supertile by simply testing whether any tile is attachable at any of the empty sites adjacent to it.
2. A supertile $C$ of size larger than the given shape can be assembled. Note that this supertile need not be terminal. We note that if any such $C$ exists, then there exists such a $C$ with size one larger than $M$. Such a $C$, along with the order in which tiles are added to assemble $C$, would suffice as a proof. We verify this proof by checking that $C$ can indeed be assembled by adding the tiles in the order specified, and that the size of the supertile is larger than the shape.

In both cases, the size of the proof is linear in the instance size, and the verification algorithm runs in time quadratic in the instance size. □

Lemma 7.3. Unq-Shape *is NP-hard under the unique shape model.*

*Proof.* The proof uses a construction proposed by Lagoudakis and LaBean [6] to solve the 3-SAT problem by exploiting the massive parallelism possible in DNA operations to emulate a nondeterministic device that solves 3-SAT.

We reduce 3-SAT to Unq-Shape using their construction. Given any instance of 3-SAT with $m$ clauses and $n$ different variables, we construct an instance of Unq-Shape, $(\langle T, \mathcal{S}, G, 2\rangle, M)$ with $|T| = O(m+n)$, and the size of the shape $M$ of order $O(mn)$. There is a one-to-one correspondence between variable assignments for the 3-SAT instance and the supertiles in $Term(\mathbb{T})$; i.e., every possible assignment to variables is associated with a distinct terminal supertile. If the assignment satisfies the formula, the terminal supertile associated with it has the shape of an $(n+2) \times (m+3)$ rectangle; else the shape associated with the assignment has the shape of an $(n+2) \times (m+3)$ rectangle with its top-right corner missing. Thus, if the tile system uniquely assembles into a rectangle with its top-right corner missing, then no assignment satisfies the 3-SAT formula; on the other hand, if the tile system does not uniquely assemble into this corner-missing rectangle, then there is at least one assignment which satisfies the 3-SAT formula. Thus, a polynomial time algorithm to decide Unq-Shape will result in a polynomial time algorithm for 3-SAT.

The idea of the reduction is to have the bottom row of the rectangle encode the clauses, have the left column encode the variables, and let the second column correspond to a possible assignment of values to the variables. The rest of the assembly evaluates the formula at that assignment, and the row below the top row encodes which of the clauses get satisfied by the assignment. A tile gets added to the top-right corner iff the assignment satisfies all the clauses.

Take any 3-SAT formula with $n$ variables, $x_1, x_2, \ldots, x_n$, appearing in $m$ clauses $C_1, C_2, \ldots, C_m$. We define a tile system as shown in Figure 6, where the temperature is fixed at 2. In addition to the seed tile, there are $n$ *variable tiles* which attach on top of the seed tile to give the first column. The east side of these tiles provides a strength-2 glue to the *assignment tiles.* Besides, there are $m$ *clause tiles* that attach to the east of the seed tile to form the bottom row. For any clause-variable pair, $C_j$ and $x_i$, the following *computation tiles* are present:

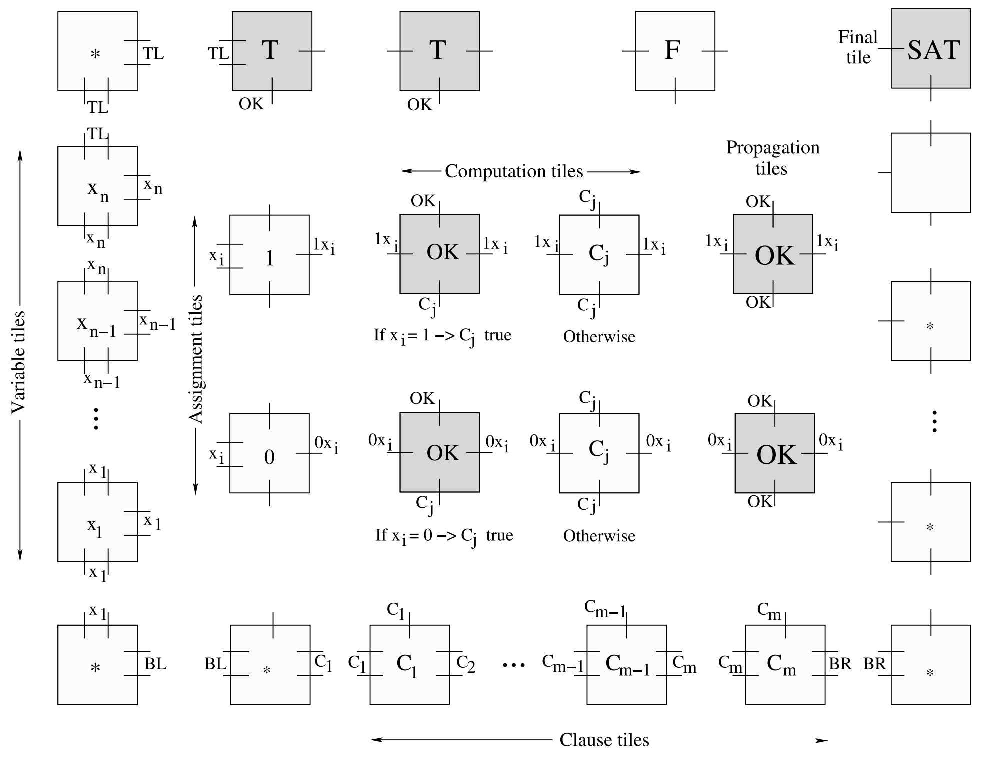


FIG. 6. *Tile system for evaluation of a* 3-*SAT formula.*

1. If $x_i$ set to 1 makes the clause $C_j$ true, then the upper-left computation tile shown in the figure is present; otherwise, the upper right one is present.
2. Similarly, if $x_i$ set to 0 makes the clause $C_j$ true, then the lower-left computation tile in the figure is present; otherwise, the lower right one is present.

In addition, there are $2n$ tiles to propagate $OK$ upwards. The tile labeled $SAT$ attaches to the top-right corner only if the formula is satisfied by the assignment in the second column of the supertile.

Figures 7(a) and (b) show two of the terminal supertiles of a tile system corresponding to the formula $(x_1 + x_2 + \overline{x_3})(x_1 + \overline{x_2} + x_3)(\overline{x_1} + x_2 + \overline{x_3})$. Note that since the assignment $x_1 = 0$, $x_2 = 1$, $x_3 = 1$ satisfies the formula, a tile gets attached in the top-right corner in Figure 7(a). Also, since the assignment $x_1 = 0$, $x_2 = 1$, $x_3 = 0$ does not satisfy the formula, the rectangle in Figure 7(b) is missing a tile in the top-right corner. $\square$

**7.2. Multiple tile and multiple temperature models.** The algorithm of [2] to determine whether a tile system uniquely assembles a given shape does not work if the self-assembly model allows preassembled supertiles to be added to the growing seed supertile. The algorithm therefore does not apply to the multiple tile model. The algorithm also assumes that once a tile is placed, it can never be removed. This is not the case in the multiple temperature model. In this section we consider the complexity of the unique shape problem under the multiple tile model and the multiple temperature model. We first show that for both models the problem is NP-hard. We then generalize the proofs of this fact to show that the problem for both models is neither in NP nor co-NP unless P = NP.

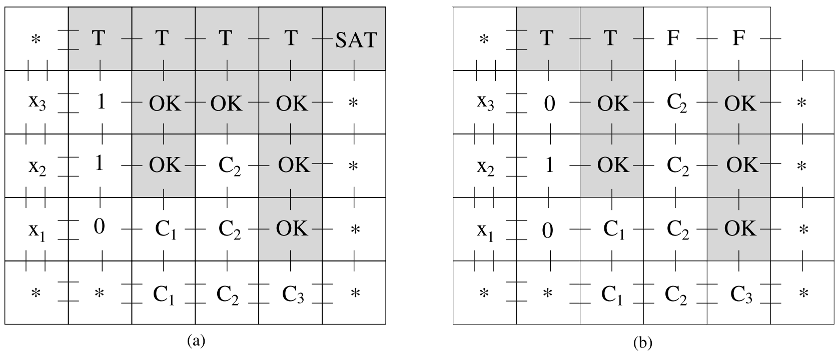


FIG. 7. *Two terminal supertiles for the formula* $(x_1 + x_2 + \overline{x_3})(x_1 + \overline{x_2} + x_3)(\overline{x_1} + x_2 + \overline{x_3})$.

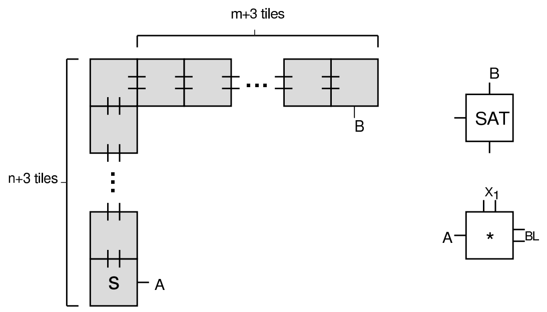


FIG. 8. *To show that the* UNQ-SHAPE *problem is NP-hard for the multiple tile model we modify the tile set of Theorem* 7.1. *We add the tiles given above as well as change the seed tile to be S, add glue A to the west edge of the old seed tile, and add glue B to the north edge of the final SAT tile.*

THEOREM 7.4. UNQ-SHAPE *is NP-hard under the multiple tile model. It remains NP-hard even when the temperature* $\tau = 2$ *and q is polynomial in the number of tiles in the system.*

*Proof.* The proof is a reduction from 3-SAT similar in spirit to the reduction in Theorem 7.1. For a given instance of 3-SAT with $n$ variables and $m$ clauses, consider a multiple tile system $\langle T, s, G, 2, q = (n+2)(m+3)\rangle$ given as follows. The tile set includes the tiles given in Figure 8, as well as the tiles for solving 3-SAT as given in Theorem 7.1, with the following adjustment. The seed tile is no longer the seed, but just a tile in the set, and it is given a west glue $A$ with glue strength 1. Also, the SAT tile from the construction is given a north glue $B$ with strength 1.

This tile set assembles a single-width chain of tiles that grows north for $n+2$ tiles and then turns east and grows for $m+3$ tiles. The east glue of the first tile in this chain (the seed tile) has east glue $A$, and the last tile in the chain has south glue $B$. Denote the shape of this supertile as $M$. Now in the case that the corresponding 3-SAT formula is not satisfiable, nothing else can be attached, and $M$ is uniquely assembled. However, if there exists a satisfying assignment to the 3-SAT formula, an $(n+2) \times (m+3)$ rectangle whose northeast-most tile has north glue $B$ and southwest-most tile has west glue $A$ is in the system's addable supertile set $W$. Thus, a supertile of the shape of an $(n+3) \times (m+4)$ rectangle can be produced. Thus, in this case $M$ is not uniquely produced. So the output of UNQ-SHAPE when given this tile system and shape $M$ as input is yes iff the corresponding 3-SAT formula is not satisfiable. Thus, UNQ-SHAPE is NP-hard under the multiple tile model. See Figure 9 for an example

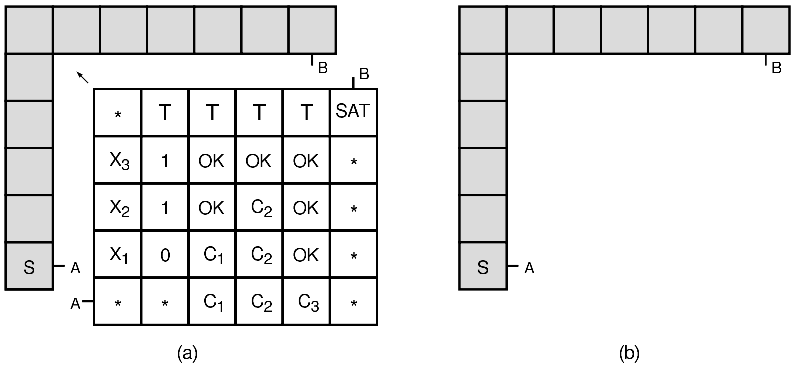


FIG. 9. *Two terminal supertiles for the formula* $(x_1 + x_2 + \overline{x_3})(x_1 + \overline{x_2} + x_3)(\overline{x_1} + x_2 + \overline{x_3})$. *In the case that a given* 3*-SAT formula is not satisfiable, the tree on the right is uniquely assembled.*

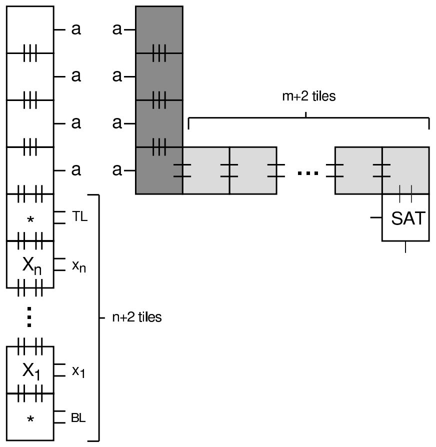


FIG. 10. *This tile set in addition to the tile set of Theorem* 7.1 *under the two-temperature model with* $\tau_1 = 2$ *and* $\tau_2 = 4$ *uniquely assembles an* $(n+6) \times 1$ *column iff the corresponding* 3*-SAT formula is unsatisfiable.*

of the construction. □

*Proof.* We again use a reduction from 3-SAT similar in spirit to the reduction in Theorem 7.1. For a given 3-SAT formula with $n$ variables and $m$ clauses, consider the two-temperature tile system with temperatures $\tau_1 = 2, \tau_2 = 4$, and tile set given by the construction in Theorem 7.1, with the additions and modifications given in Figure 10. The modifications increase the north/south glue strengths of the tiles that occur in the first column of the $(n + 2) \times (m + 3)$ construction from strength 2 to strength 4. Additionally, four tiles are added such that the length of the first column in the construction grows to height $n + 6$ rather than $n + 2$. Another modification is made to the final SAT tile such that it has a strength-2 glue on its north edge. This, in combination with other added tiles, allows a new chain of tiles to be added in the case that a satisfying set of variable assignments are given. The new tiles grow westward back to the first column of the construction. Here four tiles are placed, growing northward now, adjacent to the four *extra* tiles in the first column. These four tiles are connected to one another by strength-3 bonds. In addition, each has a west glue of strength-1. See Figure 11 for an example of the construction.

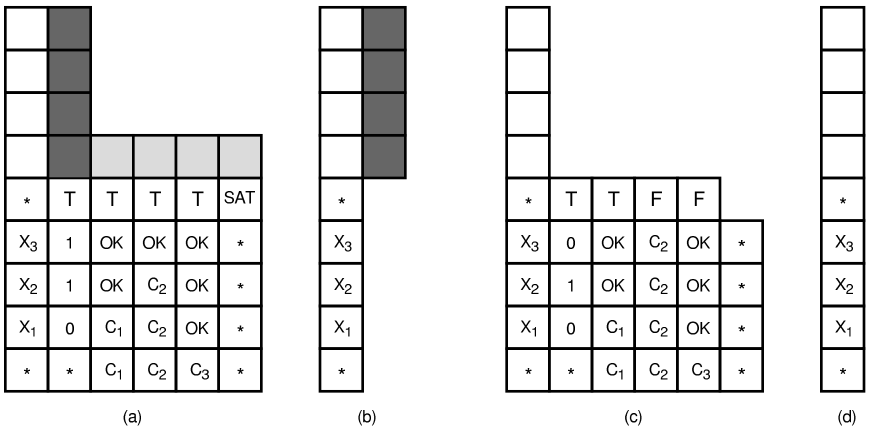


FIG. 11. *At the initial temperature $\tau_1 = 2$, a supertile of the shape given in* (a) *is formed if a satisfying assignment to the* 3*-SAT variables is given. Otherwise, a supertile of the shape given in* (c) *is constructed. At temperature $\tau_2 = 4$, supertiles like* (a) *are broken down to the supertile in* (b). *Supertiles of form* (c) *are broken down into the supertile in* (d). *The supertile in* (d) *is uniquely assembled iff the corresponding* 3*-SAT formula is unsatisfiable.*

THEOREM 7.5. UNQ-SHAPE *is NP-hard under the multiple temperature model. It remains NP-hard even when the temperature is raised only once.*

The idea is that now the temperature of the system is raised from 2 to 4. At temperature 4, each tile between column 2 and column $m + 3$, row 1 to $n + 2$, can be removed, going from bottom to top, right to left. All tiles in row $n + 3$ from column 3 to $m + 3$ can also be removed, leaving a height $n + 6$ column with four extra tiles attached at the end. Nothing within this supertile can be removed, and clearly nothing can be added, so this structure is terminal. However, in the case that no satisfying arrangement exists, the four extra tiles can never get added in the first phase of the two-temperature assembly process. Thus in this case the shape $M$ equal to an $(n + 6) \times 1$ column is uniquely assembled. Thus, when given this tile set and shape $M$ as input, the UNQ-SHAPE problem outputs yes iff the corresponding 3-SAT formula is not satisfiable. So the UNQ-SHAPE problem is NP-hard for the two-temperature model. ▯

THEOREM 7.6. UNQ-SHAPE *is not in NP or co-NP under the multiple temperature model unless $P = NP$. This remains true even when the temperature is raised only once.*

*Proof.* To show this we generalize the reduction from co-SAT given in Theorem 7.5. Our construction yields, depending on the input shape, a reduction from either SAT or co-SAT to the problem of deciding whether a given tile set assembles the given shape. The "not in co-NP" part of the theorem follows from the SAT reduction, and the "not in NP" part from the co-SAT reduction.

The reduction is as follows. Given a 3-SAT formula with $n$ variables and $m$ clauses, we create a tile set at temperature $\tau_1 = 2$ in the fashion of Theorem 3.2 that assembles an $n$-bit binary counter growing from west to east, which counts from 0 to $2^n - 1$. In this case, however, at each incrementation of the counter the given 3-SAT formula is evaluated for the variable assignment corresponding to the current value of the binary counter. After the final iteration of the counter, a string of tiles grows back to the initial column of tiles and *latches on* by creating a second vertical column that is attached adjacent to the first. However, this growth is disrupted if any occurrence

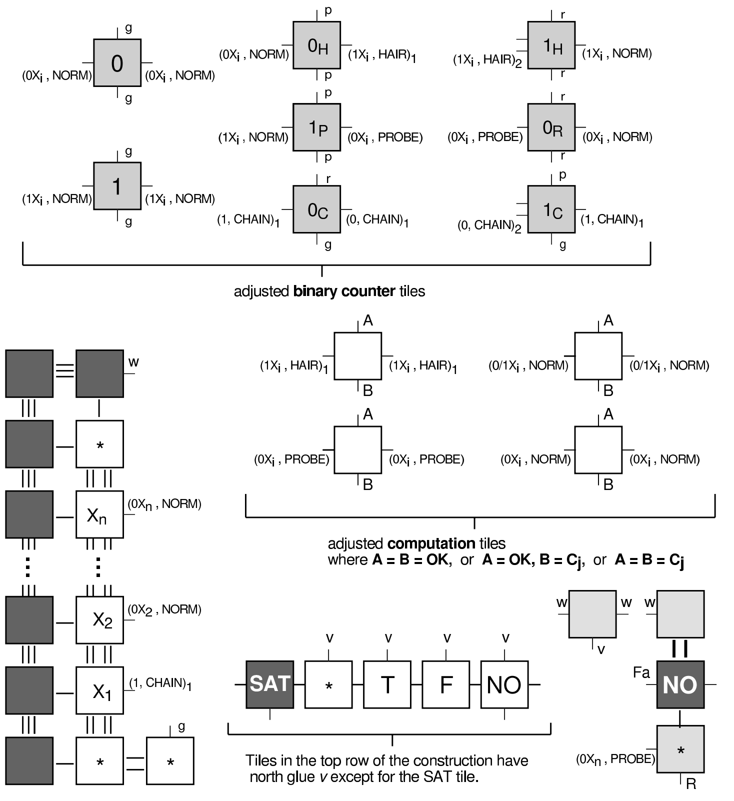

adjusted binary counter tiles
adjusted computation tiles
where A = B = OK, or A = OK, B = Cj, or A = B = Cj
Tiles in the top row of the construction have north glue v except for the SAT tile.
SAT
T
F
NO
Fa
(0Xi , NORM)
(1Xi , HAIR)1
(1Xi , PROBE)
(0Xi , PROBE)
(1, CHAIN)1
(0, CHAIN)1
(0Xn , NORM)
(0X2 , NORM)
(0Xn , PROBE)

FIG. 12. *To show that the* UNQ-SHAPE *problem is not in NP or co-NP unless P=NP under the multiple temperature model, we combine a binary counter with the SAT solving tiles of section* 7.1 *to evaluate a given SAT formula for every possible assignment to the variables. Some tiles or glues of the construction are omitted in this figure as they are unchanged from the construction in section* 7.1.

of an SAT tile is encountered. Thus, such a latch is created iff there is no satisfying assignment to the variables.

The temperature of the system is then raised to $\tau_2 = 4$. This eliminates any tile occurring east of the initial variable column. What is left is a single column supertile in the case that there exists a satisfying assignment, and a pair of adjacent columns in the case that there is no satisfying assignment. The tile set for this construction is given in Figure 12. Let shape $M$ be the single column, and shape $W$ be the double column. Thus, when given this tile set and shape $M$ as input, the UNQ-SHAPE problem outputs yes iff the corresponding 3-SAT formula is satisfiable. If shape $W$ is given as input, the UNQ-SHAPE problem outputs yes iff the corresponding 3-SAT formula is not satisfiable. The first input exhibits the reduction from SAT, the second from co-SAT. See Figure 13 for an example. □

THEOREM 7.7. UNQ-SHAPE *is not in NP or co-NP under the multiple tile model unless $P = NP$. This remains true even when the temperature $\tau = 2$.*

*Proof.* The proof is again a reduction from 3-SAT and very similar to the proof for Theorem 7.6. Consider a given instance of 3-SAT with $n$ variables and $m$ clauses. Let the size of the addable supertiles be $q = 2^n(n+3)(m+2)+2(n+3)$. The tile set for the reduction is essentially that from Figure 12 except we remove all but one of the clamping tiles, reduce all glue strengths greater than 2 down to 2, and add tiles $S_1$ and $S_2$ given in Figure 14, with $S_1$ being set as the seed tile.

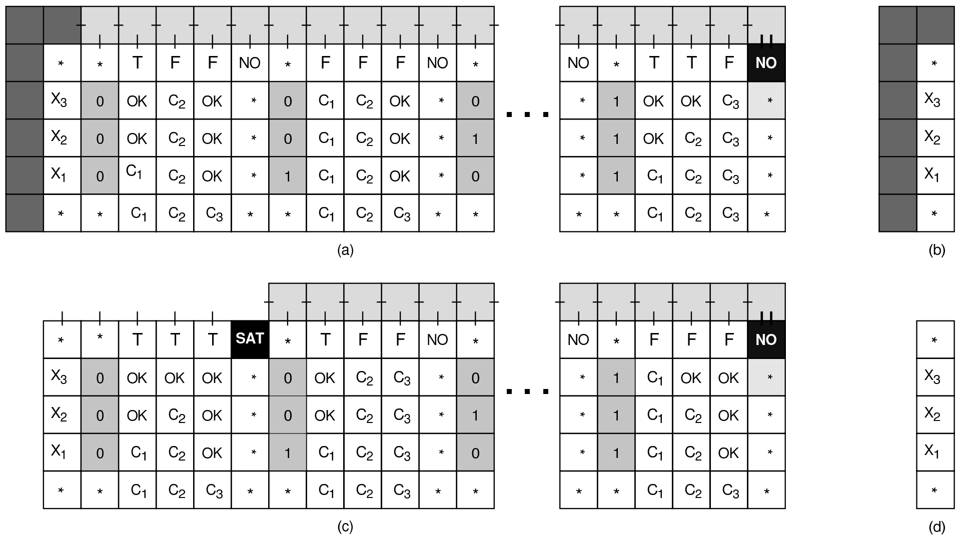


FIG. 13. *At temperature $\tau = 2$, if the corresponding SAT formula has no satisfying assignment, then a growth of tiles, starting at the final NO tile, is started and reaches back to the initial column of variable tiles and clamps on. In the case that there are one or more satisfying assignments, the growth is stopped, and a shape like* (c) *is formed. After raising the temperature to $\tau = 4$, shape* (b) *is uniquely produced in the first case and shape* (d) *in the second. Thus, the* UNQ-SHAPE *problem with the tile set of Figure* 12 *yields a reduction from co-SAT with input shape* (b) *and a reduction from SAT with input shape* (d).

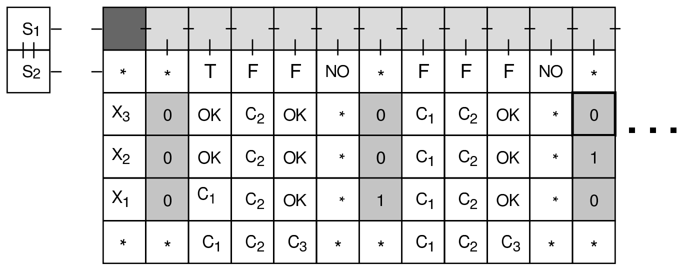


FIG. 14. *Under the multiple tile model with $q = 2^n(n+3)(m+2) + 2(n+3)$, the large $(2^n(m+2)+2) \times (n+3)$ rectangle can be attached to the two S tiles iff the corresponding* 3*-SAT formula is satisfied.*

Clearly this tile set uniquely assembles a two-tile supertile consisting only of $S_1$ and $S_2$ in the case that the 3-SAT formula is satisfiable. On the other hand, if there is no satisfying arrangement, the complete size $(2^n(m+2)+2) \times (n+3)(m+2)$ rectangle from the tiles of Figure 12 is in the set of addable supertiles. The resultant uniquely assembled shape is then the rectangle with the tiles $S_1$ and $S_2$ attached to the side. Thus, the unique shape problem with the size-2 shape given as input gives a reduction from SAT, while the rectangle with the two $S$ tiles attached gives a reduction from co-SAT. ∎

## 8. Conclusions.

We have considered various natural generalizations of the self-assembly model in an attempt to reduce the tile complexity of certain shapes. In doing so, we have produced a construction for building arbitrary-sized rectangles, which is particularly useful for the assembly of thin rectangles (a $k \times N$ rectangle is thin if $k < \frac{\log N}{\log\log N - \log\log\log N}$). This construction also yields a simple way to assemble thicker rectangles while achieving the same optimal tile and time complexity achieved in [1]. We use modifications of this construction to show how two of the models, the multitemperature model and the flexible glue model, can reduce tile complexity below theoretical lower bounds for the standard model for certain shapes. In particular, the multitemperature model assembles thin $k \times N$ rectangles in optimal $\Theta(\frac{\log N}{\log\log N})$, while the flexible glue model reduces the complexity of $\Theta(\frac{\log N}{\log\log N})$ for the assembly of $N \times N$ squares to the tight bound of $\Theta(\sqrt{\log N})$. For the remaining models, we have extended Kolmogorov lower bound proofs from [9] to show that, for many shapes, the new models do not improve tile complexity. We have also shown that the problem of verifying whether a given tile set assembles into a given shape is co-NPC under the unique shape model and not in NP or co-NP unless P = NP under the multiple tile and multiple temperature models, in contrast to the polynomial time solution for the standard and flexible glue models.

There are a number of open questions remaining, some of which are as follows. First, there are some questions as to how the parameter $q$ in the multiple tile model affects complexity. The proof of the Kolmogorov lower bound for $N \times N$ squares does not hold if the bound of $q$ on the size of the addable supertiles is removed. Also, to show that the Unq-Shape problem under the multiple tile model is neither in NP nor co-NP we required that $q$ be exponential in the size of the tile set. To show NP-hardness, we needed $q$ to grow linearly in the size of the tile set. For polynomial sized $q$, is the problem in NP or co-NP, and for constant $q$, is the problem in P?

Second, can new examples be found showing that the generalized models can reduce tile complexity? For example, are there shapes for which the multiple tile or the unique shape models reduce complexity? And for the multiple temperature model, can it help to use more than two temperatures? If so, does the temperature need only be monotonically increasing or can it help to raise and lower the temperature? Also, the construction for the flexible glue model shows how to create rectangles with width at least logarithmic in their length, but does not work for thinner rectangles. However, for *medium* rectangles (width less than $\log N$ but not thin) we can combine the flexible glue model with the two-temperature model to attain the tight $\Theta(\sqrt{\log N})$ tile complexity. Is it possible to assemble such a rectangle using only the flexible glue model? And finally, even in the case of the standard model, the Kolmogorov lower bounds for the assembly of $N \times N$ squares (Theorems 6.1, 6.2) do not apply if the temperature of the system is a large exponential function of $N$. If this is allowed, is it possible to reduce the tile complexity of assembling $N \times N$ squares?

**Acknowledgment.** We would like to thank the anonymous reviewer of this paper for the extremely detailed and helpful comments.

## REFERENCES

[1] L. Adleman, Q. Cheng, A. Goel, and M. Huang, *Running time and program size for self-assembled squares*, in Proceedings of the 33rd Annual ACM Symposium on Theory of Computing, Heraklion, Greece, 2001, ACM, New York, pp. 740–748.

[2] L. Adleman, Q. Cheng, A. Goel, M. Huang, D. Kempe, P. Espanes, and P. Rothemund, *Combinatorial optimization problems in self-assembly*, in Proceedings of the 34th Annual ACM Symposium on Theory of Computing, Montreal, Canada, 2002, ACM, New York, pp. 23–32.

[3] Q. Cheng and P. M. de Espanes, *Resolving Two Open Problems in the Self-Assembly of Squares*, Technical Report 793, University of Southern California, Los Angeles, CA, 2003.

[4] T.-J. Fu and N. C. Seeman, *DNA double-crossover molecules*, Biochem., 32 (1993), pp. 3211–3220.

[5] T. H. LaBean, H. Yan, J. Kopatsch, F. Liu, E. Winfree, H. J. Reif, and N. C. Seeman, *The construction, analysis, ligation, and self-assembly of DNA triple crossover complexes*, J. Amer. Chem. Soc., 122 (2000), pp. 1848–1860.

[6] M. G. Lagoudakis and T. H. Labean, 2*D DNA self-assembly for satisfiability*, in Proceedings of the 5th DIMACS Workshop on DNA Based Computers, Cambridge, MA, 1999, AMS, Providence, RI, pp. 459–468.

[7] M. Li and P. Vitanyi, *An Introduction to Kolmogorov Complexity and Its Applications*, 2nd ed., Springer Verlag, New York, 1997.

[8] J. H. Reif, *Local parallel biomolecular computation*, in Proceedings of the 3rd Annual DIMACS Workshop on DNA Based Computers, Philadelphia, PA, 1997, H. Rubin and D. H. Wood, eds., DIMACS Ser. Discrete Math. Theoret. Comput. Sci. 48, AMS, Providence, RI, 1999, pp. 217–254.

[9] P. Rothemund and E. Winfree, *The program-size complexity of self-assembled squares*, in Proceedings of the 32nd Annual ACM Symposium on Theory of Computing, Portland, OR, 2000, ACM, New York, pp. 459–468.

[10] H. Wang, *Proving theorems by pattern recognition*, Bell Systems Tech. J., 40 (1961), pp. 1–42.

[11] E. Winfree, *Algorithmic Self-Assembly of DNA*, Ph.D. thesis, California Institute of Technology, Pasadena, CA, 1998.

[12] E. Winfree, F. Liu, L. Wenzler, and N. Seeman, *Design and self-assembly of two-dimensional DNA crystals*, Nature, 394 (1998), pp. 539–544.

[13] E. Winfree, X. Yang, and N. C. Seeman, *Universal computation via self-assembly of DNA: Some theory and experiments*, in DNA Based Computers II, L. F. Landweber and E. B. Baum, eds., DIMACS 44, AMS, Providence, RI, 1996, pp. 191–213.

SIAM J. Comput.
Vol. 34, No. 6, pp. 1516–1528

# CONVERGENCE PROPERTIES OF THE GRAVITATIONAL ALGORITHM IN ASYNCHRONOUS ROBOT SYSTEMS[*]

REUVEN COHEN[†] AND DAVID PELEG[†]

**Abstract.** This paper considers the convergence problem in autonomous mobile robot systems. A natural algorithm for the problem requires the robots to move towards their center of gravity. This paper proves the correctness of the gravitational algorithm in the fully asynchronous model. It also analyzes its convergence rate and establishes its convergence in the presence of crash faults.

**Key words.** robot swarms, autonomous mobile robots, convergence

**AMS subject classifications.** 68Q22, 70B15

**DOI.** 10.1137/S0097539704446475

## 1. Introduction.

### 1.1. Background and motivation.

Swarms of low cost robots provide an attractive alternative when facing various large-scale tasks in hazardous or hostile environments. Such systems can be made cheaper, more flexible, and potentially resilient to malfunction. Indeed, interest in autonomous mobile robot systems arose in a variety of contexts (see [4, 5, 13, 14, 15, 16, 17, 18, 19, 20, 27] and the survey in [6, 7]).

Along with developments related to the physical engineering aspects of such robot systems, there have been recent research attempts geared at developing suitable algorithmics, particularly for handling the distributed coordination of multiple robots [3, 8, 9, 21, 23, 25, 26]. A number of computational models were proposed in the literature for multiple robot systems. In this paper we consider the fully asynchronous model of [8, 9, 11, 22]. In this model, the robots are assumed to be identical and indistinguishable, lack means of communication, and operate in Look-Compute-Move cycles. Each robot wakes up at unspecified times, observes its environment using its sensors (capable of identifying the locations of the other robots), performs a local computation determining its next move, and moves accordingly.

Much of the literature on distributed control algorithms for autonomous mobile robots has concentrated on two basic tasks, called *gathering* and *convergence*. Gathering requires the robots to occupy a single point within finite time, regardless of their initial configuration. Convergence is the closely related task in which the robots are required to converge to a single point, rather than reach it. More precisely, for every $\epsilon > 0$ there must be a time $t_\epsilon$ from which all robots are within a distance of at most $\epsilon$ of each other.

A common and straightforward approach to these tasks relies on the robots in the swarm calculating some median position and moving towards it. Arguably, the most natural variant of this approach is the one based on using the *center of gravity* (sometimes called the *center of mass*, the *barycenter*, or the average) of the robot swarm. This approach is easy to analyze in the synchronous model. In the asynchronous

[*]Received by the editors October 22, 2004; accepted for publication (in revised form) April 18, 2005; published electronically August 31, 2005. Parts of this paper appeared in preliminary form in *Proceedings of the 12th Annual European Symposium on Algorithms*, Bergen, Norway, 2004.
http://www.siam.org/journals/sicomp/34-6/44647.html

[†]Department of Computer Science and Applied Mathematics, The Weizmann Institute of Science, Rehovot 76100, Israel (r.cohen@weizmann.ac.il, david.peleg@weizmann.ac.il). The work of the first author was supported by the Pacific Theaters Foundation. The work of the second author was supported by the Israel Science Foundation (grant 693/04).

model, analyzing the process becomes more involved, since the robots operate at different rates and may take measurements at different times, including while other robots are in movement. The inherent asynchrony in operation might therefore cause various oscillatory effects on the centers of gravity calculated by the robots, preventing them from moving towards each other and possibly even causing them to diverge and stray away from each other in certain scenarios.

Several alternative, more involved, algorithms have been proposed in the literature for the gathering and convergence problems. The gathering problem was first discussed in [25, 26] in a semisynchronous model, where the robots operate in cycles but not all robots are active in every cycle. It was proven therein that it is impossible to gather *two* oblivious autonomous mobile robots without a common orientation under the semisynchronous model (although 2-robot convergence is easy to achieve in this setting). On the other hand, there is an algorithm for gathering $N \geq 3$ robots in the semisynchronous model [26]. In the asynchronous model, an algorithm for gathering $N = 3, 4$ robots is presented in [9, 22], and an algorithm for gathering $N \geq 5$ robots has recently been described in [8]. The gathering problem was also studied in a system where the robots have limited visibility [2, 12]. Fault-tolerant algorithms for gathering were studied in [1]. In a failure-prone system, a gathering algorithm is required to successfully gather the nonfaulty robots, independently of the behavior of the faulty ones. The paper presents an algorithm tolerant against a single crash failure in the asynchronous model. For Byzantine faults, it is shown therein that in the asynchronous model it is impossible to gather a 3-robot system, even in the presence of a single Byzantine fault. In the fully synchronous model, an algorithm is provided for gathering $N$ robots with up to $f$ faults, where $N \geq 3f + 1$.

Despite the existence of these elaborate gathering algorithms, the gravitational approach is still very attractive for a number of reasons. To begin with, it requires only very simple and efficient calculations, which can be performed on simple hardware with minimal computational efforts. It can be applied equally easily to any number of dimensions and to any swarm size. Moreover, the errors it incurs due to rounding are bounded and simple to calculate. In addition, it is oblivious (i.e., it does not require the robots to store any information on their previous operations or on past system configurations). This makes the method both memory-efficient and self-stabilizing (meaning that following a finite number of transient errors that change the states of some of the robots into other (possibly illegal) states, the system returns to a legal state and achieves eventual convergence). Finally, the method avoids deadlocks, in the sense that every robot can move at any given position (unless it has already reached the center of gravity). These advantages may well make the gravitational algorithm the method of choice in many practical situations.

Subsequently, it is interesting to study the correctness and complexity properties of the gravitational approach to convergence. This study is the focus of the current paper. We prove the convergence of the center of gravity algorithm in the fully asynchronous model. We also analyze the convergence rate of the algorithm. Finally, we establish convergence in the crash fault model. Specifically, we show that in the presence of $f$ crash faults, $1 \leq f \leq N - 2$, the $N - f$ nonfaulty robots will converge to the center of gravity of the crashed robots.

### 1.2. The model.

The basic model studied in [3, 8, 9, 21, 23, 25, 26] can be summarized as follows. The $N$ robots execute a given algorithm in order to achieve a prespecified task. Each robot $i$ in the system operates individually, repeatedly going through simple cycles consisting of three steps:

- *Look*: Identify the locations of all robots in $i$'s private coordinate system and obtain (instantaneously) a multiset of points $P = \{p_1, \dots, p_N\}$ defining the current *configuration*. The robots are indistinguishable, so $i$ knows its own location $p_i$ but does not know the identity of the robots at each of the other points. This model allows robots to detect multiplicities; i.e., when two or more robots reside at the same point, all robots will detect this fact. Note that this model is stronger than, e.g., the one of [8].
- *Compute*: Execute the given algorithm, resulting in a goal point $p_G$.
- *Move*: Move on a straight line towards the point $p_G$. The robot might stop before reaching its goal point $p_G$ but is guaranteed to traverse a distance of at least $S$ (unless it has reached the goal). The value of $S$ is not assumed to be known to the robots, and they cannot use it in their calculations.

This model, which allows the robot to suddenly stop short of reaching its goal point, is henceforth referred to as the *sudden-stop* model. The simpler model where the robot always reaches $p_G$ is henceforth called the *undisturbed-motion* model. For simplicity of presentation, we will prove some of our claims first in the undisturbed-motion model and then extend the proof to the full (sudden-stop) model.

The Look and Move operations are identical in every cycle, and the differences between various algorithms are in the Compute step. The procedure carried out in the Compute step is identical for all robots.

In most papers in this area (cf. [9, 12, 24, 25]), the robots are assumed to be rather limited. To begin with, the robots are assumed to have no means of directly communicating with each other. Moreover, they are assumed to be *oblivious* (or memoryless); namely, they cannot remember their previous states, their previous actions, or the previous positions of the other robots. Hence the algorithm used in the Compute step cannot rely on information from previous cycles, and its only input is the current configuration. While this is admittedly an overrestrictive and unrealistic assumption, developing algorithms for the oblivious model still makes sense in various settings for two reasons. First, solutions that rely on nonobliviousness do not necessarily work in a dynamic environment where the robots are activated in different cycles, or robots might be added to or removed from the system dynamically. Second, any algorithm that works correctly for oblivious robots is inherently self-stabilizing; i.e., it withstands transient errors that alter the robots' local states into other (possibly illegal) states.

We consider mainly the *fully asynchronous* ($\mathcal{ASYNC}$) timing model (cf. [8, 9]). In this model, robots operate on their own (time-varying) rates, and no assumptions are made regarding the relative speeds of different robots. In particular, robots may remain inactive for arbitrarily long periods between consecutive operation cycles (subject to some "fairness" assumption that ensures that each robot is activated infinitely often in an infinite execution). This feature is sometimes modeled by incorporating a *Wait* phase in each operation cycle of the robots.

We find it instructive to compare the performance of the gravitational algorithm in this model with its performance in two alternative models studied in the literature, namely, the *semisynchronous* ($\mathcal{SSYNC}$) model (cf. [25]) and the *fully synchronous* ($\mathcal{FSYNC}$) model (cf. [26]). In the fully synchronous model, all robots operate at fixed time cycles, and the Look phase of all robots is simultaneous. In the commonly studied intermediate semisynchronous model, it is again assumed that there are fixed time cycles, and the Look phase of the robots is simultaneous. However, now only a subset of the robots may wake up at every cycle.

To describe the center of gravity algorithm, hereafter named Algorithm Go_to_COG, we use the following notation. Denote by $\bar{r}_i[t]$ the location of robot $i$ at time $t$. Denote the true center of gravity at time $t$ by $\bar{c}[t] = \frac{1}{N}\sum_{i=1}^{N}\bar{r}_i[t]$. Denote by $\bar{c}_i[t]$ the center of gravity as last calculated by the robot $i$ before or at time $t$; i.e., if the last calculation by $i$ was done at time $t' \leq t$, then $\bar{c}_i[t] = \bar{c}[t']$. Note that, as mentioned before, robot $i$ calculates this location in its own private coordinate system; however, for the purpose of describing the algorithm and its analysis, it is convenient to represent these locations in a unified global coordinate system (which, of course, is unknown to the robots themselves). This is justified by the linearity of the center of gravity calculation, which renders it invariant under any linear transformation. By convention, $\bar{c}_i[0] = \bar{r}_i[0]$ for all $i$.

Algorithm Go_to_COG is very simple. After measuring the current configuration at some time $t$, the robot $i$ computes as its goal point the average location of all robot positions (including its own), $\bar{c}_i[t] = \sum_j \bar{r}_j[t]/N$, and then proceeds to move towards the calculated goal point $\bar{c}_i[t]$. A formal definition of this algorithm follows.

```
Algorithm Go_to_COG (code for robot $i$ at time $t$):
    1. Calculate the center of gravity, $\bar{c}_i[t] = \frac{1}{N}\sum_j \bar{r}_j[t]$.
    2. Move to the point $\bar{c}_i[t]$.
```

As mentioned earlier, the move may terminate before the robot $i$ actually reaches the point $\bar{c}_i[t]$. The point at which the robot does stop its movement is henceforth referred to as its *destination point.* More formally, define the destination point $\bar{\gamma}_i[t]$ of robot $i$ to be the final point of the movement made by $i$ following the last Look performed by $i$ before or at time $t$. Note that at time $t$, robot $i$ may not have computed its goal point $\bar{c}_i[t]$ yet, and, even if it had, it has no knowledge of the possibility of sudden stops; hence it is unaware of its destination point $\bar{\gamma}_i[t]$. Nevertheless, for the analysis we may treat this point as given at the moment of the Look action. Recall also that even in case the robot $i$ has not reached $\bar{c}_i[t]$, it must have traversed a distance of at least $S$.

## 2. Asynchronous convergence.

This section proves our main result, namely, that Algorithm Go_to_COG guarantees the convergence of $N$ robots for any $N \geq 2$ in the asynchronous model. Notice that in the $\mathcal{ASYNC}$ model, since no guarantees are given as to the behavior of the robot's location and velocity at the duration of the Move phase, the Compute and Wait phases may be assimilated into the Move phase and treated as time periods during the Move phase in which the robot progresses with zero velocity. Hence in the analysis we may consider only the Look and Move phases.

LEMMA 2.1. *If for some time $t_0$, $\bar{r}_i[t_0]$ and $\bar{\gamma}_i[t_0]$ for all $i$ reside in the interior of a closed convex curve, $\mathcal{P}$, then for every time $t > t_0$, $\bar{r}_i[t]$ and $\bar{\gamma}_i[t]$ also reside in the interior of $\mathcal{P}$ for every $1 \leq i \leq N$.*

*Proof.* For the Move operation, it is clear that if for some $i$, $\bar{r}_i[t_0]$ and $\bar{\gamma}_i[t_0]$ both reside in the interior of a convex $\mathcal{P}$, then for the rest of the Move operation $\bar{\gamma}_i[t] = \bar{\gamma}_i[t_0]$ does not change and $\bar{r}_i[t]$ remains on the segment $[\bar{r}_i[t_0], \bar{\gamma}_i[t_0]]$, which is inside $\mathcal{P}$.

For the Look step, the claim is obvious for the $\bar{r}_i[t]$ values. We first argue that the calculated centers of gravity $\bar{c}_i[t]$, for every $i$, also reside in the interior of $\mathcal{P}$. If $N = 2$, then the calculated center of gravity is on the line segment connecting both robots and therefore respects convexity. For $N > 2$ robots, the center of gravity is on the line connecting the center of gravity of $N - 1$ robots and the $N$th robot, and the claim follows by induction. The lemma now follows since the destination $\bar{\gamma}_i[t]$ of robot $i$ is on the line segment connecting the robot location $\bar{r}_i[t]$ and its calculated center of gravity $\bar{c}_i[t]$ and therefore also respects convexity. $\square$

Hereafter we assume, for the time being, that the configuration is one-dimensional; i.e., the robots reside on the $x$-axis. Later on, we extend the convergence proof to $d$ dimensions by applying the result to each dimension separately.

For every time $t$, let $H[t]$ denote the convex hull of the points $\{\bar{r}_i[t] \mid 1 \le i \le N\} \cup \{\bar{\gamma}_i[t] \mid 1 \le i \le N\}$, namely, the smallest closed interval containing all $2N$ points. Lemma 2.1 yields the following.

COROLLARY 2.2. *For $N \ge 2$ robots and times $t_1, t_0$, if $t_1 > t_0$, then $H[t_1] \subseteq H[t_0]$.*

Unfortunately, it is hard to prove convergence on the basis of the size of $H$ alone, since it is hard to show that it strictly decreases. Other potentially promising measures, such as $\phi_1$ and $\phi_2$ defined next, also prove problematic, as they might sometimes increase in certain scenarios. Subsequently, the measure $\psi$ we use in what follows to prove strict convergence is defined as a combination of a number of different measures. Formally, let us define the following quantities:

$$\begin{aligned}
\phi_1[t] &= \sum_{i=1}^{N} |\bar{c}[t] - \bar{\gamma}_i[t]| \ , \\
\phi_2[t] &= \sum_{i=1}^{N} |\bar{\gamma}_i[t] - \bar{r}_i[t]| \ , \\
\phi[t] &= \phi_1[t] + \phi_2[t] \ , \\
h[t] &= |H[t]| \ , \\
\psi[t] &= \frac{\phi[t]}{2N} + h[t] \ .
\end{aligned}$$

We now claim that $\phi$, $h$, and $\psi$ are nonincreasing functions of time.

LEMMA 2.3. *For every $t_1 > t_0$, $\phi[t_1] \le \phi[t_0]$.*

*Proof.* Examine the change in $\phi$ due to the various robot actions. Suppose a Look operation is performed by robot $i$ at time $t$. (If two or more robots perform a Look operation simultaneously, then we serialize these operations for the sake of analysis and consider their sequential effects.) Denote with a superscript $^b$ (respectively, $^a$) the values of the robot locations and centers of gravity and the above quantities just before (resp., after) the (instantaneous) Look operation. Then $\bar{\gamma}_i^b[t] = \bar{r}_i^b[t] = \bar{r}_i^a[t]$. Moreover, $\bar{r}_j^a[t] = \bar{r}_j^b[t]$ and $\bar{\gamma}_j^a[t] = \bar{\gamma}_j^b[t]$ for every $j \ne i$; hence also $\bar{c}^a[t] = \bar{c}^b[t]$. These equalities imply the following. First, denoting the contributions of the robots $j \ne i$ to $\phi_1$ and $\phi_2$ by $\tilde{\phi}_1[t] = \sum_{j \ne i} |\bar{c}[t] - \bar{\gamma}_j[t]|$ and $\tilde{\phi}_2[t] = \sum_{j \ne i} |\bar{\gamma}_j[t] - \bar{r}_j[t]|$, respectively, these contributions do not change; namely, $\tilde{\phi}_1^a[t] = \tilde{\phi}_1^b[t]$ and $\tilde{\phi}_2^a[t] = \tilde{\phi}_2^b[t]$. Also, the contributions of robot $i$ to $\phi_1^b$ and $\phi_2^b$ are $|\bar{c}^b[t] - \bar{\gamma}_i^b[t]| = |\bar{c}^b[t] - \bar{r}_i^b[t]|$ and $\bar{\gamma}_i^b[t] - \bar{r}_i^b[t] = 0$, respectively. Finally, the contributions of robot $i$ to $\phi_1^a$ and $\phi_2^a$ are $|\bar{r}_i^a[t] - \bar{\gamma}_i^a[t]|$ and $|\bar{\gamma}_i^a[t] - \bar{c}^a[t]|$. Hence the total contribution of robot $i$ to $\phi^a$ is the same as its contribution to $\phi^b$, since the point $\bar{\gamma}_i^a[t]$ occurs on the segment $[\bar{r}_i^b[t], \bar{c}^b[t]] = [\bar{r}_i^a[t], \bar{c}^a[t]]$. (In the undisturbed-motion case the situation is even simpler, as the robot always reaches its calculated destination and therefore $\bar{\gamma}_i^a[t] = \bar{c}^b[t]$.) Therefore, $\phi$ is unchanged by the Look performed.

Now consider some time interval $[t_0', t_1'] \subseteq [t_0, t_1]$, such that no Look operations were performed during $[t_0', t_1']$. Suppose that during this interval each robot $i$ moved a distance $\Delta_i$ (where some of these distances may be 0). Then $\phi_2$ decreased by $\sum_i \Delta_i$, the maximum change in the center of gravity is $|\bar{c}[t'_1] - \bar{c}[t'_0]| \leq \sum_i \Delta_i / N$, and the robots' calculated centers of gravity have not changed. Therefore, the change in $\phi_1$ is at most $\phi_1[t'_1] - \phi_1[t'_0] \leq \sum_i \Delta_i$. Hence the sum $\phi = \phi_1 + \phi_2$ cannot increase. $\square$

LEMMA 2.4. *$\psi$ is a nonincreasing function of time.*

*Proof.* By Lemma 2.3, $\phi$ is nonincreasing. By Corollary 2.2, $h$ is nonincreasing. Therefore their sum is also nonincreasing. $\square$

LEMMA 2.5. *For all $t$, $h[t] \leq \psi[t] \leq 2h[t]$.*

*Proof.* The lower bound is trivial. For the upper bound, notice that $\phi[t]$ is the sum of $2N$ summands, each of which is at most $h[t]$ (since they all reside in the segment).

We now state a lemma which allows the analysis of the change in $\phi[t]$ (and therefore also $\psi[t]$) in terms of the contributions of individual robots. It is, in general, impossible to separate the change in $\phi[t]$ to contributions of individual robots. However, it is possible to bound the minimum decrease in $\phi[t]$ by the decrease caused by the motion of a single robot.

LEMMA 2.6. *If $H[t_0] = [0,1]$ and at some time $t \geq t_0$ all robots are in the interval $[0, 1/2]$ (i.e., $\bar{r}_i[t] \in [0, 1/2]$ for all $i$), then there exists a time $t_1 \geq t$, such that $\psi[t_1] \leq \left(1 - \frac{1}{8N^2}\right) \psi[t_0]$.*

*Proof.* We split the situation into two subcases:

1. At time $t$ all destination points $\bar{\gamma}_i[t]$ resided in the interval $[0, \frac{3}{4}]$. In this case, take $t_1$ to be the time when each robot has completed at least one cycle. All robots and destinations are now in the interval $[0, \frac{3}{4}]$, and therefore $h \leq \frac{3}{4}$. Now $h[t_1] \leq \frac{3}{4} \leq \frac{3}{4} h[t_0]$, and therefore

$$\psi[t_1] = h[t_1] + \frac{\phi[t_1]}{2N} \leq \frac{3}{4} h[t_0] + \frac{\phi[t_0]}{2N} \leq \frac{7}{8} h[t_0] + \frac{7}{8} \frac{\phi[t_0]}{2N} = \frac{7}{8} \psi[t_0],$$

where the last inequality is due to the fact that $\frac{\phi[t_0]}{2N} \leq h[t_0]$ as argued in the proof of Lemma 2.5. The lemma immediately follows in this case.

2. At time $t$ there existed robots with $\bar{\gamma}_i[t] > \frac{3}{4}$. In this case, take $k$ to be the robot with the highest destination point (or one of them) and take $t_1$ to be the time robot $k$ completes its next Move. Its Move size is at least $\Delta_k \geq \frac{1}{4}$. Suppose at some time interval $[t', t''] \subseteq [t, t_1]$ no Look was performed by any robot and that every robot, $i$, moved in this time interval by the vector $\bar{\delta}_i$. Now, all robots approached their destinations, so $\phi_2[t''] = \phi_2[t] - \frac{1}{N} \sum_i \delta_i$. The center of gravity was changed by the robots' motions to $\bar{c}[t''] = \bar{c}[t'] + \frac{1}{N} \sum_i \bar{\delta}_i$ and therefore also $\bar{c}[t''] - \bar{c}[t'] \leq \frac{1}{N} \sum_i \delta_i$. Since no Look operations occur in the time interval $[t', t'']$, no $\bar{\gamma}_i$ is changed, and for every $i$, $|\bar{c}[t''] - \bar{\gamma}_i[t'']| \leq |\bar{c}[t''] - \bar{\gamma}_i[t'']| + \frac{1}{N} \sum_i \delta_i$. For $i = k$,

$$\begin{aligned} |\bar{c}[t''] - \bar{\gamma}_k[t'']| &= \left| \bar{c}[t'] + \frac{\bar{\delta}_k}{N} + \frac{1}{N} \sum_{i \neq k} \bar{\delta}_i - \bar{\gamma}_k[t'] \right| \\ &\leq \left| \bar{c}[t'] + \frac{\bar{\delta}_k}{N} - \bar{\gamma}_k[t'] \right| + \frac{1}{N} \sum_{i \neq k} \delta_j . \end{aligned}$$

Since robot $k$ is approaching $\bar{\gamma}_k$ from the left, it follows that $\bar{r}_k + \bar{\delta}_k \leq \bar{\gamma}_k$. Since by its maximality all robots are to the left of $\bar{\gamma}_k$ at all times, it follows that $\bar{c} = \frac{1}{N}(\sum_{i \neq k} \bar{r}_i + \bar{r}_k) \leq \bar{\gamma}_k - \frac{\delta_k}{N}$. Therefore,

$$|\bar{c}[t''] - \bar{\gamma}_k[t'']| \leq |\bar{c}[t'] - \bar{\gamma}_k[t']| + \frac{1}{N}\left(\sum_i \delta_i - 2\delta_k\right).$$

Look operations do not change $\psi$, and the total movement of robot $k$ is $\Delta_k \geq \frac{1}{4}$. Therefore, $\phi$ decreased by at least $2\Delta_k/N$, and the term $\frac{\phi}{2N}$ in $\psi$ decreased by at least $\frac{2\Delta_k/N}{2N} = \frac{\Delta_k}{N^2} \geq \frac{1}{4N^2}$. As $\psi[t_0] \leq 2$ and $h$ is nonincreasing, $\psi$ decreased by at least $\frac{1}{4N^2} \geq \frac{\psi[t_0]}{8N^2}$. The lemma follows. $\square$

We now give our main lemma. First, we treat the undisturbed-motion model. Note that in this case, the destination points are always the calculated centers of gravity.

LEMMA 2.7. *In the undisturbed-motion model, for every time $t_0$, there exists some time $\hat{t} > t_0$ such that*

$$\psi[\hat{t}] \leq \left(1 - \frac{1}{8N^2}\right)\psi[t_0].$$

*Proof.* Assume, without loss of generality, that at time $t_0$, the robots and their destination points resided in the interval $H[t_0] = [0, 1]$ (and thus $h[t_0] = 1$ and $\psi[t_0] \leq 2$). Take $t^*$ to be the earliest time after $t_0$ when each robot has completed at least one entire Look-Compute-Move cycle. There are now two possibilities.

*Case* (1). Every destination point $\bar{\gamma}_i[t']$ that was calculated at any time $t' \in [t_0, t^*]$ resided in the segment $(\frac{1}{2N}, 1]$. In this case, at time $t^*$ no robot can reside in the segment $[0, \frac{1}{2N}]$, since every robot has completed at least one cycle operation, where it has arrived at its calculated center of gravity outside the segment $[0, \frac{1}{2N}]$, and from then on it may have moved a few more times to its newly calculated centers of gravity, which were also outside this segment, by Corollary 2.2. Hence at time $\hat{t} = t^*$ all robots and centers of gravity reside in $H[\hat{t}] \subseteq [\frac{1}{2N}, 1]$, so $h[\hat{t}] \leq 1 - \frac{1}{2N}$, and $\phi[\hat{t}] \leq \phi[t_0]$. Therefore,

$$\psi[\hat{t}] \;=\; \frac{\phi[\hat{t}]}{2N} + h[\hat{t}] \;\leq\; \frac{\phi[t_0]}{2N} + 1 - \frac{1}{2N} \;=\; \psi[t_0] - \frac{1}{2N}\;.$$

Also, by Lemma 2.5, $\psi[t] \leq 2$, and hence $\frac{1}{2N} \geq \frac{1}{4N}\psi[t_0]$. Combined, we get $\psi[\hat{t}] \leq \left(1 - \frac{1}{4N}\right)\psi[t_0]$.

*Case* (2). For some $t_1 \in [t_0, t^*]$, the destination point (or center of gravity) $\bar{\gamma}_i[t_1] = \bar{c}_i[t_1] = \frac{1}{N}\sum_{j=1}^{N} \bar{r}_j[t_1]$ calculated by some robot $i$ at time $t_1$ resided in $[0, \frac{1}{2N}]$. Therefore, at time $t_1$ all robots resided in the segment $[0, \frac{1}{2}]$. The lemma then follows by applying Lemma 2.6. $\square$

To prove the convergence of the gravitational algorithm in the undisturbed-motion model in $d$-dimensional Euclidean space, we apply Lemma 2.7 to each dimension separately. Observe that by Lemmas 2.7 and 2.5, for every $\epsilon > 0$ there is a time $t_\epsilon$ by which $h[t_\epsilon] \leq \psi[t_\epsilon] \leq \epsilon$; hence the robots have converged to an $\epsilon$-neighborhood. This proves that in the undisturbed-motion model in $d$-dimensional Euclidean space, for any $N \geq 2$, $N$ robots performing Algorithm `Go_to_COG` will converge.

We now turn to the full (sudden-stop) model. In this case the following lemma replaces Lemma 2.7.

LEMMA 2.8. *In the full (sudden-stop) model in $d$ dimensions, for every time $t_0$, there exists some time $\hat{t} > t_0$ such that*

$$\psi[\hat{t}] \le \max\left\{ \left(1 - \frac{1}{8N^2}\right)\psi[t_0] \ , \ \psi[t_0] - \frac{S}{4N\sqrt{d}} \right\}.$$

*Proof.* The proof is similar to the proof of Lemma 2.7 with the following changes.

- In each round we treat only the dimension for which $h$ is largest (or one such dimension if there exist more than one). We refer to this dimension as the $x$-dimension and denote quantities in this dimension by ${}^{(x)}$.
- In Case (1), the moving robots may not complete their move and therefore may not leave the segment $[0, \frac{1}{2N}]$ even if no center of gravity was calculated in this segment. We split this case into two cases. If at the time of the last Look no robot resided in the segment $[0, \frac{1}{4N}]$, then no robot will reside there also at the end of the Move, and $h$ must have decreased by at least $\frac{1}{4N}$. Otherwise, the distance between each of the robots in the segment $[0, \frac{1}{4N}]$ to its observed center of gravity in the $x$-dimension is at least $\frac{1}{4N}$, since by assumption $\bar{c}[t] \ge \frac{1}{2N}$ for all $t \in [t_0, \hat{t}]$. Recalling that $d$ is the number of dimensions, we conclude that the total distance from these robots to the center of gravity is at most $\sqrt{d}$, since by the maximality of $h$ in dimension $x$ over every other dimension $\nu$, $h^{(\nu)}[t_0] \le h^{(x)}[t_0] = 1$. Thus, the ratio of the $x$-component of the vector $\bar{r}_i[t] - \bar{c}[t]$ to the vector size is at least $\frac{1}{4N\sqrt{d}}$. Therefore, the projection of a vector of length at least $S$ in this direction on the $x$-axis is at least $\frac{S}{4N\sqrt{d}}$, and $h^{(x)}$ decreases by at least this amount.
- Case (2) still holds, since Lemma 2.6 is true even with sudden stops. $\square$

Lemma 2.8 yields the following.

THEOREM 2.9. *In the full (sudden-stop) $\mathcal{ASYNC}$ model for any $N \ge 2$, in $d$-dimensional Euclidean space, $N$ robots performing Algorithm* `Go_to_COG` *will converge.*

*Proof.* Denote $\Psi[t] = \sum_{\nu=1}^{d} \psi^{(\nu)}[t]$. Then

$$\Psi[t_0] \ \le \ 2\sum_{\nu=1}^{d} h^{(\nu)}[t_0] \ \le \ 2dh^{(x)}[t_0] \ \le \ 2d\psi^{(x)}[t_0].$$

Therefore, the value of $\psi$ for the largest dimension is $\psi^{(x)}[t_0] \ge \Psi[t_0]/2d$. By Lemma 2.8 the value of $\psi$ for the largest dimension decreases after every two complete cycles of the robot swarm by an additive or multiplicative constant. The theorem follows. $\square$

## 3. Convergence rate.

**3. Convergence rate.** To bound the rate of convergence in the fully asynchronous model, one should make some normalizing assumption on the operational speed of the robots. A standard type of assumption is based on defining the maximum length of a robot cycle during the execution (i.e., the maximum time interval between two consecutive Look steps of the same robot) as one time unit. For our purposes it is more convenient to make the slightly modified assumption that for every time $t$, during the time interval $[t, t+1]$ every robot has completed at least one cycle. Note that the two assumptions are equivalent up to a constant factor of 2. Note also that this assumption is used only for the purpose of complexity analysis and was not used in our correctness proof.

**3.1. The undisturbed-motion model.** First, we discuss the convergence rate in the undisturbed-motion model.

LEMMA 3.1. *In the undisturbed-motion model, for every time interval* $[t_0, t_1]$,

$$\psi[t_1] \;\le\; \left(1-\frac{1}{8N^2}\right)^{\left\lfloor\frac{t_1-t_0}{2}\right\rfloor}\psi[t_0].$$

*Proof.* Consider the different cases analyzed in the proof of Lemma 2.7. By our timing assumption, we can take $t^* = t_0+1$. In Case (1), $\psi$ is decreased by a factor of $1-\frac{1}{4N}$ by time $\hat{t}=t^*$, i.e., within one time unit. In Case (2), we have two possibilities. If Case (1) of Lemma 2.6 holds, then $\psi$ is decreased by a factor of $\frac{7}{8}$ by time $\hat{t}=t^*$, i.e., within one time unit again. The slowest convergence rate is obtained in Case (2) of Lemma 2.6. Here we can take $\hat{t}=t^*+1=t_0+2$ and conclude that $\psi$ is decreased by a factor of $1-\frac{1}{8N^2}$ after two time units. The lemma follows by assuming a worst-case scenario in which during the time interval $[t_0,t_1]$, the "slow" Case (2) is repeated for $\lfloor\frac{t_1-t_0}{2}\rfloor$ times. ∎

By the two inequalities of Lemma 2.5 we have that $h[t_1]\le\psi[t_1]$ and $\psi[t_0]\le 2h[t_0]$, respectively. Lemma 3.1 now yields the following lemma.

LEMMA 3.2. *In the undisturbed-motion* $\mathcal{ASYNC}$ *model, for every time interval* $[t_0,t_1]$,

$$h[t_1] \;\le\; 2\left(1-\frac{1}{8N^2}\right)^{\left\lfloor\frac{t_1-t_0}{2}\right\rfloor}h[t_0]\;.$$

COROLLARY 3.3. *In any execution of the gravitational algorithm in the undisturbed-motion* $\mathcal{ASYNC}$ *model, over every interval of* $O(N^2)$ *time units, the size of the d-dimensional convex hull of the robot locations and centers of gravity is halved in each dimension separately.*

An example for slow convergence is given by the following lemma.

LEMMA 3.4. *There exist executions of the gravitational algorithm in which* $\Omega(N)$ *time is required to halve the convex hull of* $N$ *robots in each dimension.*

*Proof.* Initially, and throughout the execution, the $N$ robots are organized on the $x$-axis. The execution consists of phases of the following structure. Each phase takes exactly one time unit, from time $t$ to time $t+1$. At each (integral) time $t$, robot 1 is at one endpoint of the bounding segment $H[t]$, while the other $N-1$ robots are at the other endpoint of the segment. Robot 1 performs a Look at time $t$ and determines its perceived center of gravity $\bar{\gamma}_i[t]$ to reside at a distance $\frac{h[t]}{N-1}$ from the distant endpoint. Next, the other $N-1$ robots perform a long sequence of (fast) cycles, bringing them to within a distance $\epsilon$ of robot 1, for arbitrarily small $\epsilon$. Robot 1 then performs its movement to its perceived center of gravity $\bar{\gamma}_i[t]$. Hence the decrease in the size of the bounding interval during the phase is $h[t]-h[t+1]=\frac{h[t]}{N-1}+\epsilon$, or, in other words, at the end of the phase, $h[t+1]\approx(1-\frac{1}{N-1})h[t]$. It follows that $O(N)$ steps are needed to reduce the interval size to half of its original size. ∎

Note that there is still a linear gap between the upper and lower bounds on the convergence rate of the gravitational algorithm as stated in Corollary 3.3 and Lemma 3.4.

It is interesting to compare these bounds with what happens in the fully synchronous ($\mathcal{FSYNC}$) model. Here all robots operate at fixed time cycles, and the Look phase of all robots is simultaneous. In this model, after the first step all robots are at the center of gravity. Therefore, we have the following.

LEMMA 3.5. *In any execution of the gravitational algorithm in the undisturbed-motion $\mathcal{FSYNC}$ model, the robots gather after a single step.*

We now turn to the intermediate semisynchronous ($\mathcal{SSYNC}$) model. In this model there are fixed time cycles, and the Look phase of the robots is simultaneous, but only a subset of the robots may wake up at every cycle. We define, for the sake of time analysis, the notion of a (possibly long) time unit so that at every time unit, every robot is awakened at least once. (Here a "time unit" may compose of many cycles.)

LEMMA 3.6. *In any execution of the gravitational algorithm in the undisturbed-motion $\mathcal{SSYNC}$ model, over every interval of $O(N)$ time units, the size of the $d$-dimensional convex hull of the robot locations and centers of gravity is halved in each dimension separately.*

*Proof.* Again, we appeal to the proof of Lemma 2.7. However, in the semisynchronous model we have the advantage that at the beginning of the Look phase, each of the robots resides at the location of its last calculated center of gravity. This implies that Case (2) of the analysis of Lemma 2.6 is impossible. Thus, it is guaranteed that at every time step $t$, $h[t]$ is decreased by at least $\frac{h[t]}{2N}$. Therefore, we conclude that, over every interval of $O(N)$ time units, the size of the $d$-dimensional convex hull of the robot locations and centers of gravity is halved in each dimension separately. $\square$

**3.2. The full (sudden-stop) model.** We now discuss the full (sudden-stop) model. We give the analogues of the above theorems in this case.

First, we discuss the fully synchronous model $\mathcal{FSYNC}$. In the following, $H[0]$ is the convex hull of the $N$ robots at time 0 and $h[0]$ is the maximum width of $H[0]$ in any of the $d$ dimensions. We have the following lemma.

LEMMA 3.7. *In any execution of the gravitational algorithm in the full (sudden-stop) $\mathcal{FSYNC}$ model, the robots achieve gathering in at most $\lceil 4h[0]d^{3/2}/S \rceil$ time.*

*Proof.* If the distance of each robot from the center of gravity is at most $S$, then at the next step they will all gather. Suppose now that there exists at least one robot whose distance from the center of gravity is greater than $S$. Since the center of gravity is within the convex hull, the largest dimension is at least $h[0] \geq S/\sqrt{d}$. Without loss of generality, assume that the projection of the hull on the maximum width dimension is on the interval $[0, a]$ and that the projection of the center of gravity $\bar{c}[0]$ is in the interval $[\frac{a}{2}, a]$. Then in each step, $t$, every robot moves by a vector $\min\{\bar{r}_i[t] - \bar{c}[t], S' \frac{\bar{r}_i[t]-\bar{c}[t]}{|\bar{r}_i[t]-\bar{c}[t]|}\}$ for some $S' \geq S$. By assumption, $a$ is the width of the largest dimension, and therefore $a \geq |\bar{r}_i[t] - \bar{c}[t]|/\sqrt{d}$. For every robot in the interval $[0, \frac{a}{4}]$, the distance to the current center of gravity will decrease in the next step by at least $\min\{\frac{a}{4}, S\frac{a/4}{a\sqrt{d}}\} \geq \frac{S}{4\sqrt{d}}$. Thus, the width of at least one dimension decreases by at least $\frac{S}{4\sqrt{d}}$ in each step. Therefore, gathering is achieved after at most $\lceil 4h[0]d^{3/2}/S \rceil$ cycles, independently of $N$. $\square$

LEMMA 3.8. *In any execution of the gravitational algorithm in the full (sudden-stop) $\mathcal{SSYNC}$ model, over every interval of $O\left(N\left\lceil \frac{h}{S}\right\rceil\right)$ time units, the size of the $d$-dimensional convex hull of the robot locations and centers of gravity is halved in each dimension separately.*

*Proof.* As in the proof of Lemma 3.6, Case (2) of the analysis of Lemma 2.6 is impossible here. This guarantees that at every time step $t$, $h[t]$ is decreased by at least $\min\{\frac{S}{4N\sqrt{d}}, \frac{h[t]}{2N}\}$. Therefore, over every interval of $O(N\left\lceil \frac{h}{S}\right\rceil)$ time units, the size of the $d$-dimensional convex hull of the robot locations and centers of gravity is halved in each dimension separately. $\square$

Finally, we present the result for the $\mathcal{ASYNC}$ model. The proof is similar to Corollary 3.3, with the additional option of sudden stops, as discussed in the proof of Lemma 2.8. We obtain the following.

THEOREM 3.9. *In any execution of the gravitational algorithm in the full (sudden-stop) $\mathcal{ASYNC}$ model, over every interval of $O\left(N^2 + \frac{Nh}{S}\right)$ time units, the size of the d-dimensional convex hull of the robot locations and centers of gravity is halved in each dimension separately.*

## 4. Fault tolerance.

In this section we consider the behavior of the gravitational algorithm in the presence of possible robot failures.

Let us first observe that the gravitational algorithm achieves *self-stabilization* (cf. [10]) in a model allowing only *transient* failures. Such a failure causes a change in the states of some robots, possibly into illegal states. That is, a measurement, calculation, or movement error may occur, causing the robot to move to a location other than the center of gravity. In essence, self-stabilization is the requirement that if the system *quiets* at some time $t_0$ (namely, no more transient errors occur), then from time $t_0$ on, the algorithm will achieve convergence, despite starting from a potentially illegal state. Notice that Theorem 2.9 makes no assumptions about the initial positions and centers of gravity, other than that they are restricted to some finite region. It follows that, due to the oblivious nature of the robots (and hence the algorithm), the robots will converge regardless of any finite number of transient errors occurring in the early stages of the execution.

We now turn to consider the *crash* fault model. This model, presented in [1], follows the common crash (or "fail-stop") fault model in distributed computing and assumes that a robot may fail by halting. This may happen at any point in time during the robot's cycle, i.e., either during the movement towards the goal point or before it has started. Once a robot has crashed, it will remain stationary indefinitely.

In [1], it is shown that in the presence of a single crash fault, it is possible to *gather* the remaining (functioning) robots to a common point. Here we avoid the gathering requirement and settle for the weaker goal of convergence. We show that the `Go_to_COG` algorithm converges for every number of crashed robots. In fact, in a sense, convergence is easier in this setting since the crashed robots determine the final convergence point for the nonfaulty robots. We have the following.

THEOREM 4.1. *In the full (sudden-stop) $\mathcal{ASYNC}$ model, consider a swarm of $N$ robots that execute Algorithm `Go_to_COG`. If $1 \leq M \leq N-2$ robots crash during the execution, then the remaining $N-M$ robots will converge to the center of gravity of the crashed robots. Moreover, the size of the robots' convex hull is halved every $O\left(N\left\lceil\frac{h}{S}\right\rceil\right)$ time units.*

*Proof.* Let us first consider an execution of the gravitational algorithm by a swarm of $N$ robots in one dimension. Without loss of generality, assume that the crashed robots were $1, \ldots, M$ and their crashing times were $t_1 \leq \cdots \leq t_M$, respectively. Consider the behavior of the algorithm starting from time $t_M$. For the analysis, a setting in which the $M$ robots crashed at general positions $\bar{r}_1, \ldots, \bar{r}_M$ is equivalent to one in which all $M$ crashed robots are concentrated in their center of gravity $\frac{1}{M}\sum_{i=1}^{M}\bar{r}_i$. Assume, without loss of generality, that this center of gravity is at 0.

Now consider some time $t_0 \geq t_M$. Let $H[t_0] = [a, b]$ for some $a \leq 0 \leq b$. By Corollary 2.2, the robots will remain in the segment $[a, b]$ at all times $t \geq t_0$. The center of gravity calculated by any nonfaulty robot $M+1 \leq j \leq N$ at time $t \geq t_0$

will then be

$$\bar{\gamma}_j[t] \;=\; \frac{1}{N}\sum_{i=1}^{N}\bar{r}_i \;=\; \frac{1}{N}\left(M\cdot 0+\sum_{i=M+1}^{N}\bar{r}_i[t]\right).$$

Hence all centers of gravity calculated hereafter will be restricted to the segment $[a', b']$, where $a' = \frac{N-M}{N}\cdot a$ and $b' = \frac{N-M}{N}\cdot b$. Consequently, denoting by $\hat{t}$ the time by which every nonfaulty robot has completed a Look-Compute-Move cycle and no sudden stops occur, we have that $H[\hat{t}] \subseteq [a', b']$, and hence $h[\hat{t}] \le \frac{N-M}{N}\cdot h[t_0]$.

Again, the argument can be extended to any number of dimensions by considering each dimension separately. It follows that the robots converge to the point 0, namely, the center of gravity of the crashed robots.

If sudden stops do occur, then the robots are not guaranteed to reach their destination. However, they are guaranteed to travel a distance of at least $S$. Therefore, using arguments similar to the proof of Lemma 2.8, if sudden stops occur, then the size of the hull in the largest dimension is decreased by at least $\frac{S}{4N\sqrt{d}}$, leading to the theorem. $\square$

**5. Conclusions.** We have considered the properties of the gravitational algorithm for convergence of mobile robot swarms. We have shown that the algorithm guarantees convergence to a point for any number of robots $N$ and in any dimension $d$. We have shown that an appropriate quantity is halved every $O(N^2)$ steps and have shown a case where $O(N)$ steps are needed for halving the invariant. Thus, a gap still exists between our bounds on the convergence rate, which is left as an open question. We have shown analogous (and somewhat stronger) results for the simpler synchronous and semisynchronous models. We have also shown that the algorithm is resilient to crash failures of any number of robots.

**Acknowledgment.** We are grateful to the anonymous reviewers, whose comments helped clarify and improve the presentation of our results.

## REFERENCES

[1] N. Agmon and D. Peleg, *Fault-tolerant gathering algorithms for autonomous mobile robots*, in Proceedings of the Fifteenth Annual ACM-SIAM Symposium on Discrete Algorithms, 2004, pp. 1063–1071.

[2] H. Ando, Y. Oasa, I. Suzuki, and M. Yamashita, *A distributed memoryless point convergence algorithm for mobile robots with limited visibility*, IEEE Trans. Robotics and Automation, 15 (1999), pp. 818–828.

[3] H. Ando, I. Suzuki, and M. Yamashita, *Formation and agreement problems for synchronous mobile robots with limited visibility*, in Proceedings of the IEEE Symposium of Intelligent Control, 1995, pp. 453–460.

[4] T. Balch and R. Arkin, *Behavior-based formation control for multi-robot teams*, IEEE Trans. Robotics and Automation, 14 (1998), pp. 926–939.

[5] G. Beni and S. Hackwood, *Coherent swarm motion under distributed control*, in Proceedings of the International Symposium on Distributed Autonomous Robotic Systems, 1992, pp. 39–52.

[6] Y. U. Cao, A. S. Fukunaga, and A. B. Kahng, *Cooperative mobile robotics: Antecedents and directions*, Autonomous Robots, 4 (1997), pp. 7–23.

[7] Y. U. Cao, A. S. Fukunaga, A. B. Kahng, and F. Meng, *Cooperative mobile robots: Antecedents and directions*, in Proceedings of the IEEE/RSJ International Conference of Intelligent Robots and Systems, 1995, pp. 226–234.

[8] M. Cieliebak, P. Flocchini, G. Prencipe, and N. Santoro, *Solving the robots gathering problem*, in Proceedings of the 30th Annual International Colloquium on Automata, Languages, and Programming, Springer, New York, 2003, pp. 1181–1196.

[9] M. Cieliebak and G. Prencipe, *Gathering autonomous mobile robots*, in Proceedings of the 9th Annual International Colloquium on Structural Information and Communication Complexity, Carleton Scientific, Waterloo, ON, Canada, 2002, pp. 57–72.

[10] S. Dolev, *Self-Stabilization*, MIT Press, Cambridge, MA, 2000.

[11] P. Flocchini, G. Prencipe, N. Santoro, and P. Widmayer, *Hard tasks for weak robots: The role of common knowledge in pattern formation by autonomous mobile robots*, in Proceedings of the 10th Annual International Symposium on Algorithms and Computation, Springer, New York, 1999, pp. 93–102.

[12] P. Flocchini, G. Prencipe, N. Santoro, and P. Widmayer, *Gathering of autonomous mobile robots with limited visibility*, in Proceedings of the 18th Annual Symposium on Theoretical Aspects of Computer Science, Springer, New York, 2001, pp. 247–258.

[13] D. Jung, G. Cheng, and A. Zelinsky, *Experiments in realising cooperation between autonomous mobile robots*, in Proceedings of the International Symposium on Experimental Robotics, Springer, New York, 1997.

[14] Y. Kawauchi, M. Inaba, and T. Fukuda, *A principle of decision making of cellular robotic system (CEBOT)*, in Proceedings of the IEEE Conference on Robotics and Automation, 1993, pp. 833–838.

[15] M. J. Mataric, *Interaction and Intelligent Behavior*, Ph.D. thesis, MIT, Cambridge, MA, 1994.

[16] S. Murata, H. Kurokawa, and S. Kokaji, *Self-assembling machine*, in Proceedings of the IEEE Conference on Robotics and Automation, 1994, pp. 441–448.

[17] L. E. Parker, *Designing control laws for cooperative agent teams*, in Proceedings of the IEEE Conference on Robotics and Automation, 1993, pp. 582–587.

[18] L. E. Parker, *On the design of behavior-based multi-robot teams*, J. Advanced Robotics, 10 (1996), pp. 547–578.

[19] L. E. Parker and C. Touzet, *Multi-robot learning in a cooperative observation task*, Distributed Autonomous Robotic Systems, 4 (2000), pp. 391–401.

[20] L. E. Parker, C. Touzet, and F. Fernandez, *Techniques for learning in multi-robot teams*, in Robot Teams: From Diversity to Polymorphism, T. Balch and L. E. Parker, eds., A. K. Peters, Wellesley, MA, 2001, pp. 191–236.

[21] G. Prencipe, *CORDA: Distributed coordination of a set of atonomous mobile robots*, in Proceedings of the 4th Annual European Research Seminar on Advances in Distributed Systems, 2001, pp. 185–190.

[22] G. Prencipe, *Distributed Coordination of a Set of Atonomous Mobile Robots*, Ph.D. thesis, Universita Degli Studi Di Pisa, Pisa, Italy, 2002.

[23] K. Sugihara and I. Suzuki, *Distributed algorithms for formation of geometric patterns with many mobile robots*, J. Robotic Systems, 13 (1996), pp. 127–139.

[24] I. Suzuki and M. Yamashita, *Agreement on a common x-y coordinate system by a group of mobile robots*, in Intelligent Robots: Sensing, Modelling and Planning, World Scientific, Singapore, 1997, pp. 305–321.

[25] I. Suzuki and M. Yamashita, *Distributed anonymous mobile robots - formation and agreement problems*, in Proceedings of the 3rd Annual Colloquium on Structural Information and Communication Complexity, Carleton Scientific, Waterloo, ON, Canada, 1996, pp. 313–330.

[26] I. Suzuki and M. Yamashita, *Distributed anonymous mobile robots: Formation of geometric patterns*, SIAM J. Comput., 28 (1999), pp. 1347–1363.

[27] I. A. Wagner and A. M. Bruckstein, *From ants to a(ge)nts*, Ann. Math. Artif. Intell., 31 (1996), pp. 1–5.